(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: 0 534 466 A2

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 92116474.5

(22) Date of filing: 25.09.92

(51) Int. Cl.5: G06F 15/40, G06F 15/403

(30) Priority: 27.09.91 US 767230

(43) Date of publication of application:
31.03.93 Bulletin 93/13

(84) Designated Contracting States:
AT BE CH DE DK ES FR GB GR IE IT LI LU MC NL PT SE

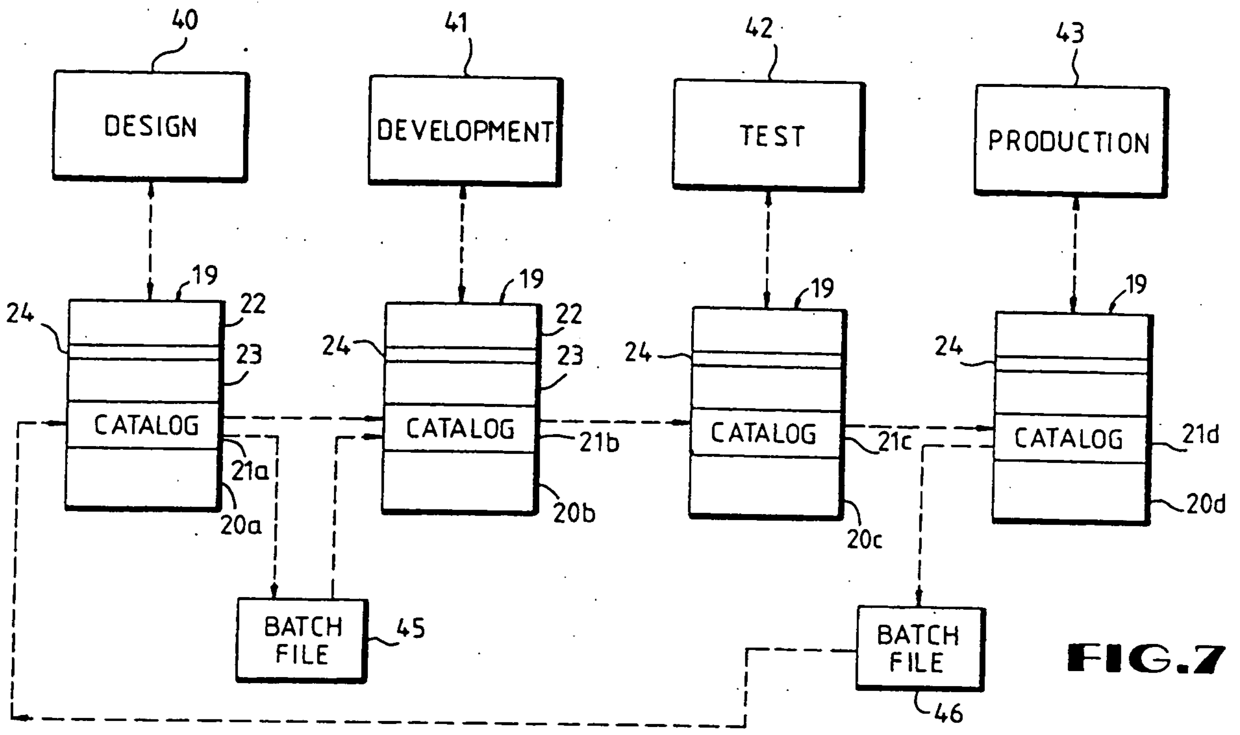
(71) Applicant: BMC Software, Inc.
1 Sugar Creek Boulevard Center Suite 500
Sugarland Texas 77478(US)

(72) Inventor: Olson, Jack E.
9800 Vista View Drive
Austin, Texas 78750(US)
Inventor: Elliott, Linda C.
1602 Springer Lane
Austin, Texas 78758(US)

(74) Representative: Altenburg, Udo, Dipl.-Phys. et al
Patent- und Rechtsanwälte
Bardehle-Pagenberg-Dost-Altenburg
Frohwitter-Geissler & Partner Postfach 86 06 20
W-8000 München 86 (DE)

(54) Change definition language for computer database system.

(57) A database application implemented on a computer includes a generic database management product (software) such as IBM DB2 along with a catalog defining the way the data itself is stored. The catalog is a definition of the tables, indexes, views, user authorizations, etc., that specify a user's particular application of the database management system. Access to the database via the catalog uses a structured query language or (SQL) which provides a way of expressing statements in a high-level language so the user will not be burdened with writing code to access the data itself. The structured query language provides statements for defining tables, indexes, views, etc., to be incorporated into the catalog. A database application (to fit a user's business) is generated and updated in a number of phases, such as design, development, test and production, and in each one of these phases a facility exists for making alterations in the database definition (catalog), all of which make use of SQL to implement the changes. According to a feature of the invention, a change definition language (CDL) is provided which is an extension of (and in the general format of) the structured query language. The change definition language allows all important alterations to be described, as changes to an existing definition, for example, and may be used by all phases of the development cycle. The CDL statements do not make the changes directly in the catalog, but instead work through SQL and another intermediate mechanism such as DB2 ALTER tailored to make changes using SQL. The changes expressed in CDL may be migrated to downstream phases and fed back to earlier phases by use of a batch of change statements expressed in CDL.



40
41
42
43
DESIGN
DEVELOPMENT
TEST
PRODUCTION
19
22
23
24
CATALOG
21a
20a
21b
20b
21c
20c
21d
20d
BATCH FILE
45
46

FIG.7

This invention relates to computer database systems, and more particularly to a language for a method of making changes to a definition of a relational database system.

A database system operating in an environment such as that provided by IBM's DATABASE 2 (DB2) computer software system commonly takes the form of a relational database product which appears to a user as a set of user-defined tables. A table is a set of columns and rows, where each column has a user-selected name and datatype, and each row is a record of data values entered for the columns. This type of database product, when fully developed and operating on a computer, includes a catalog (or dictionary) which defines these tables, as well as indexes and views of the tables, and relationships between them, plus the data itself entered by users. A structured query language, referred to as SQL, is employed to access the data, and is also used to define the form of the database, i.e., describe the tables, and describe indexes and views of the tables and other objects of the database. This structured query language is a high level programming language specifically designed for the database product, and permits a user to access or define elements without resort to a lower-level language or assembly. The statements of SQL are limited, however, and so other programming tools and languages are used, particularly in design and development phases of application definition.

After such a database system has been initially developed (e.g., by a DB2 customer's in-house programmers), the system is frequently subjected to an on-going series of changes as it is upgraded, debugged, expanded, etc. A commercially-available product useful for generating these changes is a product called "DB2 ALTER," sold by BMC Software, Inc., the assignee of this invention; this product functions to allow a user to describe changes in an interactive way (at a terminal), then these changes are implemented by using SQL to make changes in the database definition itself. Generally, a product such as DB2 ALTER effects changes by producing a sequence of operations expressed in SQL and other functional languages for unloading a database, wiping out (dropping) a part of the catalog, rebuilding a new part of the catalog to replace that which was dropped, then restructuring the data according to the revised catalog. Commercially available products providing features analogous to DB2 ALTER include RC MIGRATE, by Platinum Technology of Lombard, Illinois, PROALTER PLUS by On-Line Software of Fort Lee, New Jersey, TRANSRELATE by Compuware, and a product made by Goal Systems International of Columbus, Ohio.

A database system for a particular application may be created and debugged by separate teams of programmers in a large organization, and these teams may use incompatible tools. A design team makes an initial design, often employing a computer-aided software engineering (CASE) tool, (for example, a commercially-available Bachman tool) and the database definition is passed on in the form of a set of SQL statements to a development team which has the responsibility of generating a production version of the database. The development team, in changing the definition for more fully tailoring the application to the customer's intent, may employ a different tool, such as DB2 ALTER, or another programming language, for its own efficiency or convenience. The developers may then turn the revised version of the database over to a testing facility, where testing and debugging result in additional changes; these changes may be implemented again in DB2 ALTER. Finally, the database is released to production use, and the team which maintains the software for the production operation may make performance-enhancing changes in the way the data is physically stored, and other changes as a result of factors discovered by data-to-day users entering data or evaluating reports generated by the database system. Meanwhile, the design team is generating updated versions of the system, adding features and incorporating changes as a result of changes in the business, hardware additions, etc., and these updated versions pass through the same series of phases - development, test and production. The downstream phases must employ the updated, revised version of the system, but yet will want to keep the revisions they have made in the previous version. All of the changes made at any phase must therefore be migrated forward to downstream users, and often also fed back to earlier phases, and this must be done in an efficient manner.

Since the design and development people are often using design tools and languages in generating the database definition, which may be different from that of the production-level programmers, and the definition and changes must be implemented in a specific mechanism such as SQL for IBM DB2, a problem has existed in passing changes from design and development to production, and integrating production-level changes into a design and development environment. Only by extensive hand-entry of change lists, and comparisons of lengthy descriptions, have these types of interaction been achievable.

In accordance with one embodiment of the invention, a database application implemented on a computer includes a generic database management product (software) along with a catalog defining the way the data itself is stored. In a particular embodiment, the database system operates in the IBM DB2 environment. The catalog is a definition of the tables, indexes, views, user authorizations, etc., that specify a user's particular application of the database management system. Access to the database via the catalog uses a structured query language. A structured query language (SQL) provides a way of expressing statements in a high-level language so the user will not be burdened with writing code to access the data itself. The structured query language provides statements for defining tables, indexes, views, etc., to be incorporated into the catalog. A database application (to fit a user's business) is generated and updated in a number of phases, such as design, development, test and production, and in each one of these phases a facility exists for making alterations in the database definition (catalog), all of which make use of SQL to implement the changes. According to a feature of the invention, a change definition language (CDL) is provided which is an extension of (and in the general format of) the structured query language. The change definition language allows all important alterations to be described, as changes to an existing definition, for example, and may be used by all phases of the development cycle. The CDL statements do not make the changes directly in the catalog, but instead work through SQL and another intermediate mechanism such as DB2 ALTER tailored to make changes using SQL. The changes expressed in CDL may be migrated to downstream phases and fed back to earlier phases by use of a batch of change statements expressed in CDL.

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as other features and advantages thereof, will be best understood by reference to the detailed description of specific embodiments which follows, when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a diagram of tables, indexes and views in a simplified example of a database;

Figure 2 is a map of memory containing a database system such as would include the application example of Figure 1;

Figure 3 is a diagram of tables in the catalog of a DB2 database for a database application;

Figure 4 is a diagram of a computer system which may be used for implementing a database management system such as that of Figures 1 and 2;

Figure 5 is a diagram of a design, development, test and production cycle of a database application;

Figure 6 is a map of data structures stored by a computer system having a DB2 database application therein;

Figure 7 is a diagram like Figure 5 of phases in a database application, showing migrating forward and feed back of batches of change statements made in a change definition language according to one embodiment of the invention;

Figure 8 is a diagram representing the steps in a method of generating changes in a database catalog using the change definition language of the invention;

Figure 9 is a diagram representing the steps in a method of generating changes in a database catalog using the methods of the prior art;

Figures 10 and 11 are a diagram of a method of implementing changes in a database system using the CDL of the invention; and

Figure 12 is a logic flow chart of a simplified method implemented in the change manager of Figure 11.

RELATIONAL DATABASE SYSTEM

A relational database product is a software database that appears to the user as if it were a number of tables, such as the tables 10, 11 and 12 of Figure 1, with each table including a field that is also used in another table (i.e., "related" to another table). A table 10 has rows 13 and columns 14, where each row is a "record" and each column is a "field" of this record, so all records in a table have the same fields or columns. There can be an arbitrary number of records (rows) in a table within the constraints of available storage space, and usually the rows are in the order that the data is entered rather than in some ordered sequence. In this simplified example database of Figure 1 the purpose is to keep track of employees and work projects for a small company, and the table 10 has the name "Employees" and contains columns for first name, last name, employee number, address, etc. The table 11 is called "Projects" and has columns 14 for Project Number, Manager (identified by employee number), Department, and Location. The table 12 is called Work__Week and for each employee records what project number he worked on in the week ending on the date indicated in the Week column. In a relational database a table may have a key field 15 (sometimes specified as it must be unique to a reçord, i.e., no two records in table 10 can have the same value for this key field 15). In this example, the employee number column is the key field 15 in table 10, and project number can be a key field in table 11. An employee number will appear more than once in the table 12, indeed once every week, so the Employee__No column cannot be a key column if specified as a unique key column. In table 11, the Project__No column is a key field, and in this table each project number should be used only once.

In order to speed up the task of finding a given record, it is possible to create indexes of the data in the tables, so if the tables are spread over a large number of pages in the storage system, the value being sought can be found without searching through the entire mass of data without direction. For example, an index 16 as seen in Figure 1 contains only the employee number (from table 10) and a pointer 17 to the actual physical page number where this record is stored. Thus, to retrieve a record for a specified employee, it is merely necessary to retrieve the index 16, then retrieve the page pointed to by the pointer 17, rather than blindly retrieving all that pages containing the data of the table 10 and searching for the specified employee. Tables such as the table 16 can be created for other columns, such as last names, SSNs, etc., as may be useful.

Another characteristic of this type of relational database product is a "view." A view is an alternative way of looking at the data in one or more tables. An example of a view 18 called Address_List is shown in Figure 1, where the last name, first name and address columns from table 10 are extracted and appear as columns of the view. Thus, a view has rows and columns just as a table, but a table is physically stored as such, while a view is not. A view is generated from the data of the stored table when its use is invoked. A view may be used to control access to a table; access to a view can be granted without granting access to the table itself, so the view shows only portions of the data in the table, screening out sensitive data, e.g., the pay column, etc. Or a view can be used to combine data from two or more tables.

Of course, the example of Figure 1 is grossly simplified; a typical implementation of a database such as DB2 on a large mainframe may have many thousands of tables, indexes and views, and the number of records (rows) in some tables may be in the hundreds of thousands or millions. Generally, the database products of interest here include those capable of maintaining the largest of database applications, such as nationally-based credit card accounting and verification systems, or the like.

Referring to Figure 2, a database system is thus stored in storage 19 as a block of data 20 which includes the actual values in the rows 13 of all the tables, plus the values in all of entries in the indexes 16. This data 20 is stored in physical storage in pages, partitioned in ways that are important from a performance standpoint. In addition, a catalog 21 is stored which is the definition of all of the tables (name of table, name of columns, specification of data types in columns), a definition of all of the indexes (name, columns, pointers), a definition of all of the views, and the other objects of the system, all this itself being in the form of a set of tables. That is, in the catalog 21 there is standardized set of tables, including a table of tables, a table of columns, a table of indexes, and a table of views, etc., as discussed below. Thus, to create or to change a database definition, the standard tables of the catalog are filled in and updated. Also stored in memory as illustrated in Figure 2 is the database program 22 itself (e.g., IBM DB2), which defines how the catalog 21 is set up and how the data 20 is accessed using the catalog. As part of the database program 22, or as a separate module, there is a mechanism for controlling user access to the database and catalog for both defining the database and then for accessing it, referred to herein as a structured query language or SQL, in a block 23 of Figure 2. As will be later described, an additional module 24 may be included, according to the invention, to provide a change mechanism (using as input a set of statements in a change definition language) to be used along with the SQL module 23 to make changes in the catalog 21 and thus alter the database definition.

Referring to Figure 3, when operating with the IBM DB2 product as referred to herein, a SYSTABLES table 25 in the catalog 21 contains one row for each table, view or alias, and the columns in this table 25 include the name (NAME) of the table, view or alias, the authorization ID (CREATOR) of the owner of the table, view or alias, the type of object (TYPE) as a table, view or alias, the database name (DBNAME), the tablespace name (TSNAME), the number of columns in a table or view (COLCOUNT), the number of relationships (e.g., tables) in which the table is dependent (PARENTS), the number of relationships in which the table is a parent (CHILDREN), and various other columns as set forth in the detailed specification of the product. A SYSCOLUMNS table 26 in the catalog 21 is another of the principal tables, set up by the database program 22 to record the columns used in any of the tables or views; this records (in columns) the name of the column (NAME), the name of the table or view containing the column (TBNAME), the authorization ID (TBCREATOR) of the owner of the table or view that contains the column, the column number or numerical place (COLNO) in the table or view, the datatype of the column (LENGTH), and various other columns as set forth in the detailed specification. Another table 27 referred to as the SYSDATABASE TABLE in DB2 contains one row for each database set up by the present instance of the DBMS, and records the name and creator in columns, as well as the storage group and bufferpool which are related to physical storage and retrieval, as will be described. Another pair of important tables are the SYSDBAUTH table 28 and the SYSTABAUTH table 29, which record the grantor and grantee for any privileges granted, the database or table name for which granted, the time and date, and details of what privilege was granted. A SYSINDEXES table 30 and a SYSVIEWS table 31 record the indexes and views record the values apparent from Figure 3 for indexes and views. Other tables are included in the catalog as explained in the DB2 product specifications. The change mechanism 24 of Figure 2 is for the purpose of generating statements for changing the set of catalog tables such as that of Figure 3, using as an intermediary a data definition language of the SQL module 23.

A database product 22 is created and sold to customers as a generic product, a DBMS or database management system, useful for an unlimited number of different applications. The customer then builds a customized database to fit its particular business by creating the tables, indexes and views, defined in the catalog 21, using the SQL. Referring to Figure 4, the customer may be running a computer system having a CPU 35 and a number of terminals 36, with memory 37 and disk storage 38, as an example. The database system used by the customer includes the purchased database product 22 (e.g., DB2) which is the generic software, along with an applications generator product 23, such as SQL or structured query language as will be described, used to produce a customized definition of how the database is to be utilized locally, i.e., define the tables of Figure 1 using the catalog tables of Figure 3. This customized definition is usually created by first writing a description in a high-level language (other than SQL) using a CASE tool to generate a file of SQL statements, then using the database program 22 to interpret the SQL statements and thus generate the catalog 21 itself. The actual data to be recorded and manipulated is stored in the data file 20, which is raw data in a format described in the catalog 21.

The development cycle of a database application such as that of Figures 1-4 is illustrated in Figure 5. The database "schema" or layout as to be defined in the catalog 21 is generated by a designer in design phase 40; the designer is skilled in systems analysis and in the programming languages used, e.g. the applications generator or SQL 23 or other language(s) used, and would usually be somewhat removed from the actual production use of the product in its final form in the field. For example, the designer need not be concerned with performance tuning factors, which are dependent upon the hardware configuration; indeed, this tuning may differ in various locations where the ultimate database application is used. The design created in the design phase 40 would be an instance of the database of Figure 2, shown in Figure 5 as a catalog 21a, and perhaps a stub or test version of the data 20a itself may be included, but this is not necessarily the real-world data to be eventually entered. A copy 22a of the database software 22 is available in the design phase, of course, and, if appropriate, a copy 23a of the applications-generator language, e.g., SQL, but often the designers use other tools for defining the application. According to the invention, as will be described below, the change mechanism 24 may be present in the representation of Figure 5, but the design-development sequence is described in general first. The database as defined by the designers (composed principally of the catalog 21a) is then transmitted to the development phase 41, where there are applications developers who are skilled programmers familiar with the applications programming language 23 (or other tool for defining the database) and in the customer's requirements in using the system. These people expand the definition in catalog 21a and recompile it to obtain the more complete definition in catalog 21b. As before, the data 20b is not yet necessarily the customer's real data as will ultimately be used, but is of the same format. The application then goes to the test phase 42, where all aspects are tested by a number of skilled users; this is a debugging phase, sometimes referred to a beta test. After the testing phase, when the application is deemed to be ready for release, it appears as another instance 21c of the catalog, and this goes to a production phase 43. There may be a number of different instances of the production phase 43, at different physical locations. Here, the data 20 of the database as described above is generated in complete and real-world form, and continuously updated on a day-to-day basis. In production, the database is used on-line by the ultimate users, which include the naive users who merely enter data, as well as skilled programmers who are maintaining the application, making corrections related to day-to-day problems surfacing from use on the line, and performance-tuning to achieve optimum speed and ease-of-use. After a period of use, the database is thus defined by another instance 21d of the catalog, reflecting changes made due to problems encountered or new knowledge gained in production use.

Thus, at each of these stages, the design phase 40, the applications phase 41, the test phase 42 and the production phase 43, a complete set of the components of the application exist. That is, the generic database software 22, the applications development language 23, the dictionary or catalog 21 and the database data file 20 (in some test form, at least) all exist, and the skilled workers at each stage may make changes in the definition of the database application as it is expressed in the catalog module 21. A separate version 21a, 21b, 21c and 21d of the definition dictionary or catalog 21 is maintained at each stage 40, 41, 42 and 43, so that a stable version is available for use at each phase. During the initial creation of the application, and during the life of the application, many changes may be made at each stage. The designers at stage 40 will continue to upgrade the application to add new features and capabilities, the developers at stage 41 will continue to implement the updated design received from the design phase and make changes as needed, the test stage 42 will make changes dictated by testing, and the production phase 43 (or multiple instances of production phase) will continue to make changes required by day-to-day problems and varying working conditions. The designers working on updates need to take into account problems found in testing or production. It is thus desired that the changes made at any stage be migrated to succeeding stages, as well as fed back to previous stages. That is, after the initial design has been passed downstream, it is not a desirable solution to merely pass along to downstream stages a revised and updated version of the application catalog 21, e.g., a catalog 21b from development phase 41 to test phase 42, because the test phase 42 will have implemented their own changes to produce an instance 21c, including changes that have not been fed back to the development phase, so these would be lost if instance 21b were adopted.

The problem of making changes in the database definition and updating all instances of the database is compounded by the fact that the original definition in the design phase 40 (and perhaps development phase 41 as well) may be done using a high-level language other than the application definition language 23 (SQL for DB2). For example, a CASE (computer aided software engineering) tool may be used by the designers in phase 40 to generate the definition of the tables of Figure 1 for the catalog 21, and this CASE tool generates a set of SQL statements for passing along to downstream phases. Subsequent changes made downstream in the development cycle, e.g., in the production phase, will be made in SQL, i.e., the built-in applications-generator language 23. These changes are not likely to be readily combinable. The SQL changes from downstream are not capable of incorporation into the CASE language definition generated in the design phase. More importantly, the downstream instances of the catalog 21 are no longer the same, after a time period where local changes have been made, and even if a list of changes is passed downstream for hand entry, the then existing local catalog 21 may not be compatible with the changes requested by the list.

According to the invention, a change definition language (used for an input CDL file for the change mechanism module 24 of Figure 5) is provided, and this language may be used by all phases of the development and production cycle. This language makes use of the data definition language of SQL, and is indeed in a format very similar to SQL statements. Unlike SQL, however, the change definition language provides a full capability of defining and making changes in all objects of the catalog 21. As will appear, SQL alone was not suitable for describing some of the most important operations.

Some characteristics of SQL and DB2 will first be described, then the change definition language will be described in more detail.

STRUCTURED QUERY LANGUAGE

The structured query language that is a part of DB2 has two major components, these being the DDL or data definition language and the DML or data manipulation language. The DDL is used for creating the database description or catalog, and for making changes in the catalog after it exists; the DDL does no actually make changes in parts of the catalog, but instead "drops" or cancels an object and rebuilds it.

The tables are created in SQL by a CREATE TABLE statement, which is of the structure defined exactly in the referenced publications, but is generally of the form illustrated in the following example for creating the Employees table 10 of Figure 1:

```
CREATE TABLE EMPLOYEES
 ( FNAME              datatype  NOT NULL,
   LNAME              datatype  NOT NULL,
   EMPLOYEE_NO        datatype  NOT NULL,
   ADDRESS            datatype,
   SUPERVISOR         datatype,
   DEPT               datatype,
   PAY                datatype,
   SSN                datatype  NOT NULL )
```

where the "datatype" is a specification of the type of data that must be entered here, e.g., a string of alphanumeric characters of specified length, a floating point numerical value of specified length, a date, etc. The NOT NULL statement means a value must be entered for this column.

A view in SQL, as discussed above in reference to Figure 1, is a single table that is derived from other tables or from other views. A view usually does not exist in physical form in the data 20, but is a virtual table instead of one that is actually stored in the database. This limits the possible update operations that can be applied to a view. A view is a way of specifying a table that is to be referenced frequently, even though it may not physically exist. For example, from the tables of Figure 1, it may be desired to retrieve the names of employees working on a project, so a statement is employed:

```
CREATE VIEW PROJECT-WEEK
  AS   SELECT    FNAME, LNAME, PROJ_NAME
          FROM      EMPLOYEES,PROJECTS
```

When a view is not needed any more, a DROP VIEW command is used to dispose of it:

```
DROP VIEW PROJECT_WEEK
```

Views can be updated by an UPDATE command, which actually updates the values in the data 20 in the underlying tables from which the view is created.

An important part of a relational database is the index. An index is an access structure or path that is specified on one or more columns of a table, making accessing rows more efficient. Executing a query will take less time if some attributes involved in the query conditions were indexed; the effect is noticeable if the number of rows is quite large. The command to generate an index of the last name column for employees from the Employees table of Figure 1 is:

```
CREATE INDEX LAST_NAMES
  ON   EMPLOYEES (LNAME)
```

The actual index would have for each employee last name a pointer to the row in the Employees table for this instance. Transparent to the user, the pointer specifies the page in the virtual memory system where this row is physically located, so it is not necessary to retrieve all pages from disk to search for a particular employee. There is a time saving in execution, of course, only if there are a large number of pages needed to contain the Employees table, i.e., a large number of employees. If the table was contained on one or two pages of memory, then the recovery time would be increased instead of decreased by using a table, because it would be necessary to first recover the index itself, then retrieve the page based on the pointer found in the index. If the number of pages of employees is large, however, creating indexes on often-used reference attributes such a last name, employee number, or SSN provides a noticeable performance improvement.

It is necessary to provide security and access privileges in some manner, and SQL does this with GRANT and REVOKE statements. Some users may be prohibited from accessing certain data, typically salary data, for example, in the Employees table of Figure 1. Some commands are allowed to be executed only by certain classes of users. A naive user, entering data or transactions at a terminal, may have access to no SQL commands at all, but instead uses the database only by means of applications programs written by the designers. A database manager may need to have access to all data and certain SQL commands, but not all, whereas a systems administrator may need a different set of access rights. A system designer of course needs access to all commands. To this end, a DBMS has an authorization subsystem which enforces these restrictions. User accounts are created, and a person needing access to a database must have an account, with account number and password, and a user must login using valid account number and password, administered by the authorization subsystem, under control of a privileged user such as the systems administrator. The systems administrator can issue a command

```
GRANT CREATETAB
   TO Acct1
```

which grants to the user having account number Acct1 the right to create tables (CREATETAB). i.e., to execute the CREATE TABLE command. This also grants to user Acct1 the ownership of the tables created, and the right to say who has rights to use the tables. Then this user issues commands to create the tables of Figure 1, and also issues the command

```
GRANT INSERT,DELETE
  ON EMPLOYEES
  TO Acct2
```

which gives the user Acct2 the right to insert and delete rows in the Employees table. Only if a clause WITH GRANT OPTION is added can the user Acct2 propagate the right to other users.

A relational database system relies on key values to constrain the relationship between information in related tables. When a column is designated a key value, then its value must be unique; no two rows can have the same value for a key column. For example, in Figure 1, the employee number would be a key, so no two employees can have the same number. The create index command is used to designate a column as a key, as follows:

```
CREATE UNIQUE INDEX    EMPLOYEE_NO_INDEX
  ON   EMPLOYEES (EMPLOYEE_NO)
```

which generates an index called Employee_No_Index that requires each employee number to be unique. The department numbers and SSNs can be likewise declared unique, so there can be no duplications.

The SQL language is a special-purpose high-level programming language that frees the database user from working with the actual physical data structures used in the database. The commands generated by the user of SQL may be employed interactively (in real time) at a terminal, or for "canned" inquires or transactions are compiled by the SQL module or a compiler to generate code for defining the data structures for storage and retrieval. SQL statements can be used in conjunction with a general-purpose high-level programming language such as C, Pascal, Cobol or PL/I, for example, or with assembly language, in which case the other programming language is the host language. An SQL statement, e.g., a data definition, query, update, view definition, or index creation, can be embedded in a host language program, using some specified designator to separate the SQL statements from the host language code. For example, in PL/I the keywords EXEC SQL precede an embedded SQL statement. In some cases SQL statements can be passed as parameters in procedure calls. The names given to columns and tables are then declared as variables in the host language, and the actual physical data of the database manipulated in this way.

When a table, index or view is created using the SQL statements discussed above, the tables of Figure 3 in the catalog 21 are changed to reflect the appropriate information.

The other part of SQL, as mentioned above, is the DML or data manipulation language, used for entering data and accessing the data in the database data section 20. The DML is not used in connection with the CDL of the invention. A DBMS using SQL has one basic statement in DML for retrieving data from a database, and that is the SELECT statement. The exact format of this statement is given in the referenced publications for IBM DB2. However, in general, a SELECT statement specifies the table from which the data is to be recovered, the columns to be recovered, and the conditions, a Boolean expression that identifies the rows to be retrieved. For example, the statement:

```
SELECT ADDRESS
  FROM EMPLOYEES
  WHERE FNAME = JOHN AND LNAME = SMITH
```

selects the address column from the table 10 called Employees in Figure 1, for the row having first name John and last name Smith. In a similar manner, the statement:

```
SELECT *


FROM EMPLOYEES
WHERE SUPERVISOR = 045
```

selects a list of all columns for the Employees table 10 having a supervisor whose employee number is "045." SELECT statements can be nested, such as:

```
SELECT LNAME, FNAME
  FROM EMPLOYEES
  WHERE ( SELECT PROJECT_NO
          FROM PROJECTS
          WHERE MANAGER_NO = 073 )
```

selects last name and first name columns from the Employee table for employees having a Project No. where the project manager (selected from the Projects table) has an Employee No. = 073.

The DML part of SQL includes three commands for modifying (entering data into) a database, these being INSERT, DELETE and UPDATE. In simple form, INSERT is used to add a row to a table, and is of the general form:

```
INSERT INTO EMPLOYEES
  VALUES ("James", "Brown", "092", "125 Maple St.
          Houston TX 77039", "053", "007",
          "12.75", "754-75-7566")
```

which would insert values for all columns of a new row in the Employees table of Figure 1. If only some but not all of the values are known, the insert would be of the form:

```
INSERT INTO EMPLOYEES (FNAME,LNAME,SSN)
  VALUES ("James","Brown","754-75-7566")
```

The delete statement is of a form very similar to the insert statement, but deletes a row or rows from a table:

```
DELETE FROM EMPLOYEES
  WHERE LNAME = "Brown"
```

would delete all rows having a employee's last name "Brown." The update command is used to modify column values of one or more selected rows, where a SET clause specifies the column or columns to be modified and their new values:

```
UPDATE EMPLOYEES
  SET ADDRESS = "321 Westview"
  WHERE EMPLOYEE_NO = 093
```

DATABASE 2 (IBM DB2)

The database management system (DBMS) product for which the change definition language of one embodiment of the invention was specifically designed is DB2 or Database 2, a commercially available product of IBM Corporation, for use with the MVS operating system. Database 2 or DB2 is described in a number of publications of the IBM Corporation, particularly the so-called "IBM Database 2 Version 2 Release 2 Library" which is a collection of ten books including titles such as "Application Programming and SQL Guide, SC26-4377-1." The DB2 database product is a relational database environment, and a specific implementation of the query language (discussed in general above) called SQL or Structured Query Language is used to access data in the DB2 database product using DML, and to create or alter a database definition using DDL. The principle of SQL here, as in other implementations of SQL, is that a single statement directed to DB2 may be used to select, create, or otherwise operate on the data, rather than requiring a user to code a sequence of instructions explaining how to access the data. SQL provides full definition and data manipulation capabilities which can be used to define objects such as tables, indexes, views, etc., (there are eleven principal objects) as well as retrieving, inserting, updating, deleting data, and controlling access authorization to data. The eleven principal objects in DB2 are database, table, view, index, storage group, tablespace, synonym, alias, authorization, plan and foreign key.

The actual physical storage in any large DBMS implementation is necessarily partitioned in some way due to the system configuration, memory size, page size, construction of secondary storage, etc. In DB2, the database is a collection of logically related objects, i.e., the physically-stored tables and indexes. The terminology used in DB2 to describe the partitioned areas of storage includes terms such as "storage group," "tablespace," "indexspace," "bufferpool," etc. Tablespace refers to the part of secondary storage where tables are stored and indexspace refers to the part where indexes are stored. A page is the unit used in data transfer between primary storage or memory 37 and secondary storage or disk 38, and a "space" is a dynamically-extendable collection of pages. Referring to Figure 6, the secondary storage of the computer system of Figure 4 may contain, in one example, a user database called DBX (that is, the data 20 for a created database having the name DBX) and another called DBY. Within a database is defined one or more tablespaces, such as the tablespace XA and tablespace XB seen in the Figure. Within each tablespace are the data areas for stored tables (one or more), shown as tables XA1-XAn and XB1-XBn. A storage group is a collection of direct-access storage areas from the same device type (e.g., a disk 38); there is no one-to-one correspondence between a database such as DBX and a storage group, so in Figure 6 it is seen that the same storage group contains all of database DBX and part of DBY. Views do no occupy corresponding storage, and can be defined over multiple tables from different databases, so no storage area in Figure 6 is allocated to views. Bufferpool is a term referring to main memory 37 area reserved to satisfy the buffering requirements for one or more tables or indexes.

In DB2, the GRANT and REVOKE statements in SQL determine specific operations granted to or revoked from a user, and are in several categories, such as table and view privileges applying to existing tables, database privileges applying to creating tables, storage privileges dealing with the use of storage objects such as tablespaces and storage groups, and system privileges applying to operations such as creating a new database. There are certain bundled privileges specific to DB2, referring to assortments of privileges. These include the system administrator or SYSADM privilege which is the highest-order privilege and includes all possible operations within the system. A database administrator DBADM privilege on a specific database allows the holder to execute any operation on that database. A database control DBACTRL privilege on a specific database is similar to DBADM except that only control operation and no data manipulation as in SQL are allowed. The database maintenance DBMAINT privilege is a subset of the DBACTRL privilege and allows the holder to execute read-only maintenance operation such as back-up on the database. The system operator privilege allows the holder to perform only console operator functions with no access to the database. The design and development phases discussed to above with reference to Figure 5 thus require SYSADM privilege level as it is necessary to make changes to all of the tables of Figure 3.

ALTERING A DATABASE CATALOG USING STANDARD SQL

There are statements in the DDL part of SQL that allow certain alterations to be made in a database definition as it is specified in the catalog 21, after the tables, indexes, views and other objects of the database have been initially defined. These are statements such as ALTER TABLE, ALTER INDEX, and DROP (TABLE, VIEW, etc.). For example, a statement:

```
ALTER TABLE EMPLOYEES
  ADD ZIPCODE datatype
```

would cause a column named ZIPCODE to be added as the last column of the Employees table of Figure 1, using a datatype as specified. This table will have the same owner and the same name as before. The detailed specifications for the ALTER TABLE statement and other statements for altering elements of a database using standard SQL/DDL are given in a pp. 100-121 of the publication "IBM Database 2 Version 2, SQL REFERENCE Release 2" available from IBM Corporation as item number SC26-4380, which is a part of the Library mentioned above.

Similarly, an ALTER INDEX statement in the existing SQL specification is available to make limited changes for an existing index. For example, the statement:

```
ALTER INDEX EMPLOYEE_NO
  CLOSE YES
```

specifies that the data sets for the index are closed when the number of processes using the index becomes zero.

The syntax diagrams used in SQL statement definitions have been described in publications such as SQL REFERENCE and elsewhere in the literature (and in Appendix A). The syntax diagrams are read from left to right and top to bottom following the path of the lines. The beginning is a double arrowhead on the left pointing to the right, and the end is a pair of arrowheads on the right pointing to one another. A single arrowhead on the right of a line indicates the statement is continued on the next line, and on the left indicates a statement is continued form the previous line. Required items appear on the horizontal line (the main path), and optional items appear below the main path. If two or more items may be chosen, they appear in a stack, and if one must be chosen then one item of the stack appears in the main path. An arrow returning to the left, above the main line, indicates an item that can be repeated separated by a space, while if the repeat arrow contains a comma the repeat items are separated by a comma. A repeat arrow above a stack indicates that more than one choice can be made from the stacked items, or a single choice repeated. Statements and keywords appear in uppercase and variables (user-supplied) appear in lowercase; a default parameter appears in boldface underscored. Sometimes a single variable represents a set of several parameters, in which case the variable parameter block (in lower-case boldface) is shown as a diagram following the end of the main path; the variable parameter block may be replaced by any of the interpretations of such a diagram.

The changes that may be made in a database definition using standard SQL/DDL statements such as ALTER TABLE are seen to be very limited. For example, there is no facility for changing the owner in any of the ALTER statements; the only way to change the owner is to create a new table or other object by the user who is to be the owner. And, there is no capability for anyone other than the creator/owner of a table to make changes in a table. Likewise, there is no facility for changing the name of a database, a table, an index or a view. A very important shortcoming is the inability to change columns in a table except to add a column; there is no facility for changing a column name or datatype, or moving a column to a particular place in a table, or changing the NULL/NOT NULL specification, for example. To make these types of changes, it has been the practice to drop a table and create a new table with the desired specification, and along with this it is necessary to generate a lengthy sequence of steps in a language other than DDL because the existing data for the previously-defined table must be preserved and transferred to the new table.

For example, to change the length of a column, the actual steps generated by the BMC ALTER program mentioned above, using DDL (without the CDL concepts of the invention) would include:

```
UNLOAD data from data 20, change length in unloaded
     data
DROP table
CREATE TABLE
LABEL ON table columns
GRANT authorizations on table
CREATE SYNONYMS
CREATE INDEXes
CREATE VIEWs that use table
CREATE VIEWs that use views
LABEL ON view columns
GRANT authorizations to views
LOAD data
RUNSTATS on table spaces and indexes
IMAGECOPY table space
Precompile, compile, link applications programs
BIND application plans
```

Of these steps, the DROP, CREATE, LABEL ON, and GRANT statements are DDL statements, but the other steps are coded by other languages.

Thus, the BMC DB2 ALTER product mentioned above operates by using SQL/DDL in combination with non-DDL code to change the data structure and the definition of the data structure of a database. In the example above, a column width may be changed by using data accessing steps to save the data, a DDL DROP statement to cancel a table containing the column, then rebuilding the table with a different-sized column using CREATE, and then reloading the saved data into the new data structure.

Due to these limitations in the change or alter operations permitted by standard SQL/DDL, it has been inefficient to use SQL alone as the language for making changes in all phases during the design, development, test and production phases of a database application. Particularly, the constraints on the standard SQL language have made changes to the catalog 21, particularly the tables of Figure 3, awkward and unreliable for the early phases of the cycle, especially when migration of the changes to downstream phases, and feedback to earlier phases, is needed.

Similarly, the BMC ALTER product mentioned above, while it greatly expands the facility for making changes in an maintaining a database application, still requires all changes to be individually entered by a programmer at a terminal, interactively, and does not account for the situation where the local instance of the catalog has been changed so it is different from that in an earlier instance.

CHANGE DEFINITION LANGUAGE

According to one embodiment of the invention, interpreter apparatus for a change definition language (CDL) is provided in the form of a program instruction for a preexisting programmable processor, so that a user can write statements to make changes in an existing database to accommodate a wide variety of commonly-needed alterations in almost all objects of the catalog or definition of the database. This change definition language is defined by the specification of the permitted statements set forth in Appendix A.

The details of one implementation of the change definition language of this embodiment are described in Appendix B, which is a BNF grammer definition with Action C routines, used as an input into a LEX/YACC parser tool commercially available from MKS (Mortice Kern Systems); parsing a group of CDL statements and transforming them into machine-executable form is a matter of routine for those of ordinary skill having the benefit of this disclosure.

The change definition language of this embodiment is written in a style very similar to SQL, so that a programmer skilled in using SQL, particularly the DDL part of SQL, will be able to use CDL with a minimum of instruction. The full SQL facility 23 is needed, for CDL to operate, i.e., SQL/DDL must be present for reference when interpreting, analyzing and executing a set of CDL-defined changes, or must be present if a change statement is entered interactively. The statements of the change definition language are indeed implemented using the underlying SQL/DDL statements and code.

Referring to the statements of the embodiment of the change definition language of Appendix A, an ALTER TABLE statement is available which is greatly expanded in capability, compared to that of the standard SQL as set forth in the SQL REFERENCE publication. For example, to change the Address column in the Employees table of Figure 1 to allow a more lengthy address to be recorded, and to change the name of this column to Home_Address, a statement would be used as follows:

```
ALTER TABLE Accntl.EMPLOYEES
  ALTER COLUMN ADDRESS new_datatype
  ALTER COLUMN ADDRESS NAME HOME_ADDRESS
```

To add a column named Zipcode to the Employees table, positioned after the Address column (instead of at the end), the following statement would be used:

```
ALTER TABLE Accntl.EMPLOYEES
  ADD COLUMN ZIPCODE AFTER ADDRESS datatype
```

To move the Employee_No column to the beginning, so it is the first column, the following statement is used:

```
ALTER TABLE Accntl.EMPLOYEES
  ALTER COLUMN ADDRESS MOVE BEFORE LNAME
```

The owner of a table is changed from user "Accnt1" to user "Accnt2" by an ALTER TABLE statement as follows:

```
ALTER TABLE Accnt1.EMPLOYEES
  OWNER Accnt2
```

The capabilities of the ALTER INDEX statement of Appendix A is also greatly expanded compared to that of standard SQL. The owner of an index is changed from user "Accnt1" to user "Accnt2" by an ALTER INDEX statement as follows:

```
ALTER INDEX Accnt1.EMPLOYEE_NO
  OWNER Accnt2
```

An index can be changed from unique to not unique or vice versa; for example, an index of the SSNs from the SSN column of the Employees table of Figure 1 is made unique by a statement:

```
ALTER INDEX Accnt1.SSN_INDEX
  UNIQUE YES
```

The name of the Employee__No index of Figure 1 is changed to Empl__Number by the statement:

```
ALTER INDEX Accnt1.EMPLOYEE_NO
  NAME EMPL_NUMBER
```

The ALTER VIEW statement of Appendix A finds no counterpart in the standard SQL/DDL specification. A user may change the owner or name of a view, add a column, change a column name (to agree with a changed table), drop a column from the view, etc. For example, to change the name of the view PROJECT__WEEK of Figure 1 to WEEK, and to add the employee number column from the Employees table and drop the employee's first name column FNAME, the following statement is used:

```
ALTER VIEW Acct1.PROJECT_WEEK
  NAME WEEK
  DROP COLUMN FNAME
  AS SELECT EMPLOYEE_NO
                    FROM EMPLOYEES
```

Another capability of the change definition language of Appendix A, as compared to the Alter statements in standard SQL/DDL, is that of defining the ownership of a database, a table, an index or a view, as being a user other than the current user. The CREATE DATABASE, CREATE TABLE and CREATE VIEW statements of Appendix A all have an alternative of changing the owner to one other than the issuer of the statement.

By using the statements of the change definition language, in any of the phases of Figure 5, the catalog 21 can be changed to describe the altered database by making changes to the tables of Figure 3. Referring to Figure 7, one way of implementing the migration and feedback is to create in one phase a batch file 45 consisting of all of the changes (expressed in change definition language) made since the original design catalog 21a was passed to the development phase. This batch file of CDL statements is executed on the catalog 21b, which itself has been changed by the development people, so it is not the same as catalog 21a. Thus, there may be inconsistencies. A characteristic of a program for executing the changes defined by the CDL of the invention is that when a change is attempted using an ALTER TABLE statement, for example, and an inconsistency is found (e.g., the column name has been locally changed), the ALTER statement will not execute but instead an error will be signalled and the changes can be reconciled by direct coding (e.g., the columns names changed to be consistent). Similarly, a batch file 46 of all of the changes made by the production phase (since the catalog 21c was received from the test phase) may be created and fed back to the design team, and this batch of statements executed on the latest catalog version 21a. Again, any inconsistencies will not execute and can be reconciled.

Referring to Figure 8, it is noted that one typical way of using the CDL according to an embodiment of the invention is for the user to interact with a CASE tool 50 or other facility such as the BMC DB2 ALTER program which is commercially available, to generate a file 51 of statements in CDL. That is, the user does not write the series of CDL statements describing the changes he wishes to make in the existing database catalog, but instead the user inputs information into a computer-aided generator 50 which interprets the user input and writes the statements of the CDL file 51. The CASE tool or DB2 ALTER program 50 may be the same tool used by a designer in defining the data structure of a database, and is merely for the convenience of the user, since the CDL file 51 could also be created directly by the user. The CDL file 51 is processed by a parser 52 to import the CDL statements to change definition tables 53, in DB2 format (i.e., expanding the CDL statements into complete SQL-compatible sequences needed to effect the desired changes). The file 53 is processed by an analyzer 54 which generates a worklist 55 of changes which take into account dependencies and the like, and reconcile differences in names, etc., so that an executable worklist is provided. The old catalog 21b (for example) is then processed by the ALTER program 56 to generate the new catalog 21c.

Referring to Figure 9, the same change operation as in Figure 8 is illustrated using prior art methods and facilities. It is seen that the processors 54 and 56 are prior art functions used in the previous methods as well as in conjunction with the CDL of the invention as represented in Figure 8. In the prior art method of Figure 9, however, the user interacted using the ALTER specification 57 (the user writing change orders) to generate the change definition tables 53. The difference between the method of Figure 8 and that of Figure 9 is that in the method according to this embodiment of the invention as seen in Figure 8 the user generates the change statements of file 51 (via the CASE tool 50 if desired) in a CDL language, which is subsequently translated into the change definition in SQL/DDL form.

Referring to Figure 10, the operation of a change method using the CDL according to the invention, is illustrated. The database software 22 is maintaining a catalog 21 for a database have data store 20. Access to the database is via SQL 23 which includes DDL and DML. The change mechanism 24 includes the commercially-available BMC DB2 ALTER program 24a, which provides an interface for defining changes in the manner of Figure 9. That is, an interactive user interface 57 provided by ALTER program 24 allows a user to enter changes, for example, by merely typing over a column name or datatype in a structured display of the table on a screen for a given table in the database previously defined. The ALTER program 24a generates first a description of the change expressed in DDL, and this is a series of statements as discussed above (UNLOAD, DROP, etc.), referred to as the list 53 of Figures 8 and 9.

In addition to the facilities provided by the ALTER program 24a, the change mechanism 24 includes a change manager 24b which has the ability to accept a change definition (list 51 of Figure 8) expressed in CDL. This is in addition to the other facilities of ALTER program 24a, rather than in place of. The change manager 24b parses the CDL list 51 and thus generates the change list 53 of Figures 8 or 9. Note that the changes described in CDL in the list 51 are of the same types as could be generated by hand using the interface 57 for the ALTER program 24a. The difference is, the CDL list 51 is machine readable, as well as being of generally SQL format and readable by a programmer familiar with SQL with little additional training.

The source of the CDL list 51 could be a user who hand-codes the list, but more likely it is generated by machine. For example, in a preferred embodiment, the CDL list 51 is generated by the facilities seen in Figure 11, where another copy of the database software 22 is maintaining another instance of the catalog 21, referred to here as 21a. This instance of Figure 11 could be the design phase 40 referred to above, for example. The access and definition mechanism or SQL 23 is present, and another instance of the change mechanism 24, including BMC ALTER and change manager. Changes could be made in the catalog 21a by a user engaging ALTER in an interactive mode as discussed above, or by entering change statements in CDL, or by use of a CASE tool 59 such as Bachman. After a series of changes have been made in the catalog 21a, the CDL list 51 is generated by the change manager 24b.

An important feature of the method as depicted in Figures 10 and 11 is communicating the changes in a very precise set of statements as permitted by CDL, rather than the drop and rebuild type of definitions permitted by DDL. That is, DDL describes a change in terms of how the resultant structure is supposed to be rebuilt, rather than how an existing structure is to be changed. If a column is to be changed in a table, DDL statements to accomplish this will include a CREATE TABLE statement naming every column in the table, whereas a CDL statement names only the changed column. Further, the CDL list 51 is a list of changes to a specified instance of the catalog 21. So, before generating the changes by the mechanism of Figure 11, the first step is to make a copy of the existing instance of the catalog (before this set of changes is made), and to this end a copy V1 of catalog 21 is made, then after the series of changes are implemented locally, another instance V2 is available. When the list 51 is generated to be sent to a downstream change manager, the way this is done is to compare version V1 with version V2 and generate change statements to define the differences; this could be done by maintaining a journal of the changes made and replaying it, but since there can be changes to changed parts, and items may be changed and then reinstated to their original form, it is more precise to generate the list 51 by comparing V1 and V2.

Referring to Figure 12, this operation of generating the change list 51 as performed by the change manager module 24b of Figure 11 is illustrated in flow chart form. A loop depicted by block 60 indicates waiting for a change session to begin. When entering a change session, the first step indicated by block 61 is to copy the existing catalog 21 to storage, so a reference is available when later the CDL list 51 is to be generated. The copy is referred to as version-1 or V1. Next a loop 62 is entered where the system waits for the user to enter a change, e.g., interactively at a terminal using BMC ALTER or using the CASE tool 59, and when a change is entered it is checked for compatibility at block 63, i.e., there is a check to see if a table exists by that name, a column by that name, etc. If not compatible, an error is displayed at block 64 and the user is given the alternative of reconciling the difference, at block 65. If not, control returns to loop 62, or, if so, the reconciliation is entered at 66. The compatible change is then checked at 67 for dependencies (a column being changed used in other tables, indexes or views, etc.), and these are reconciled at block 68. After, consistencies and dependencies are all checked and defined, the requested change is implemented at block 69, i.e., the catalog V2 is altered to implement the requested change and its associated necessary changes. The session can continue if needed, at decision point 70, so additional changes are entered as long as the user wishes. When continue point 70 indicates "no," the output list 51 is generated as seen in Figure x by first loading V1 and V2, blocks 72 and 73, and comparing the two, block 74. If they are different, a list of the differences is generated at block 75, and this list is converted to CDL language at block 76, as by a parser, and sent out as the CDL list 51.

While this invention has been described with reference to specific embodiments, this description is not meant to be construed in a limiting sense. Various modifications of the disclosed embodiments, as well as other embodiments of the invention, will be apparent to persons skilled in the art upon reference to this description. It is therefore contemplated that the appended claims will cover any such modifications or embodiments as fall within the true scope of the invention.

APPENDIX A

About This Manual

This manual defines the syntax and functionality of CHANGE DEFINITION LANGUAGE (CDL), an extension of SQL's Data Definition Language (DDL). Where DDL provides only limited support for database modifications, CDL provides access to almost all of DB2's features

This manual assumes a thorough understanding of DB2 data structures and the Data Definition Language (DDL) of SQL. It is intended for advanced users who will be reviewing CDL, or for software developers who will be writing programs that can either read or write CDL. Since CDL is an extension of DDL, this manual assumes a familiarity with the terms and concepts presented in the *IBM SQL Language Reference*.

Since CDL is intended for import into BMC CHANGE MANAGEMENT *for DB2*, this manual includes an appendix on the format of CHANGE MANAGEMENT's CDL input files.

Conventions

Several special elements were used in this manual to make it easier to use. They are as follows:

Note:
A note contains important information.

Warning!
Warnings alert you to situations that could cause problems for you, like loss of data, if you do not follow the instructions carefully.

Syntax Diagram Elements

The syntax diagrams in this manual present language statements from left to right, top to bottom, along a primary path line similar to the following:

```
▶▶─STATEMENT ─parameter ──────────────────▶◀
```

The ▶▶ symbol indicates the start of a diagram, while the ▶◀ symbol terminates a diagram. Required items (such as STATEMENT and parameter in the diagram above) appear on the main track; optional items appear below the main track, as shown by option in the diagram below:

```
▶▶─STATEMENT ─┬──────────────────────────▶◀
              └─option ─┘
```

Arrows are used to continue statements from one line to the next; the solid arrowhead symbol ▶ at the end of a line indicates that the statement is continued onto the next line, while a ▶ symbol at the beginning of a line indicates that the line is a continuation of the one above:

```
▶▶─STATEMENT ─option ──────────────────────▶
▶─additional options ──────────────────────▶◀
```

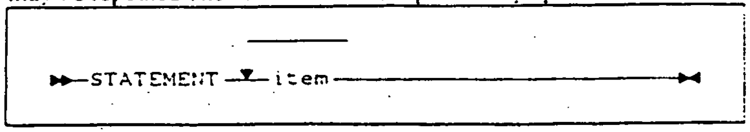
Multiple lines indicate that one path must be selected from two or more possible paths. If at least one selection is required, then one of the options will appear on the main track:

```
▶▶─STATEMENT ─┬─option_1 ─┬─────────────────▶◀
              └─option_2 ─┘
```

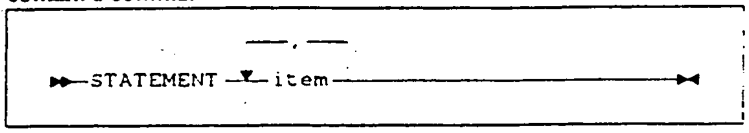
If all paths are optional, then they will all appear below the main track:

```
▶▶─STATEMENT ─┬───────────┬─────────────────▶◀
              ├─option_1 ─┤
              └─option_2 ─┘
```

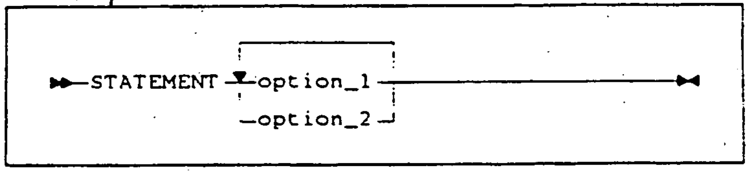
Vertical Arrows above the main track indicate that one or more syntactical elements may be repeated. In the following example, item may be repeated one or more times, separated by spaces:



If the items must be separated by commas, the repeat arrow will contain a comma:

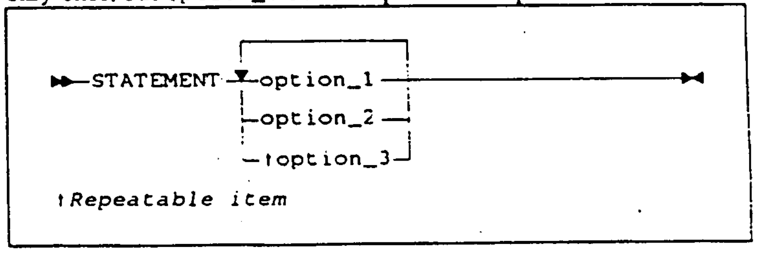


Multiple Selections are indicated by a combination of a repeat arrow and multiple selection tracks. For example, the following diagram shows that one or more paths may be selected from the list of available paths:



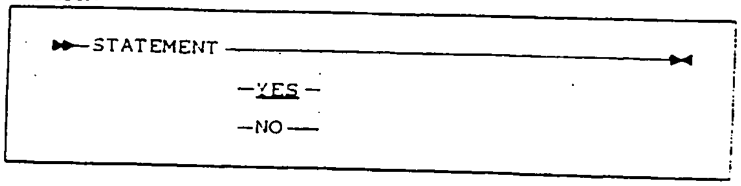
Usually, each possible path can be selected once and only once. However, in some cases, an path may be repeated (usually with varying parameters). Statements that allow paths to be repeated more than once will be noted in the description of the specific statement where applicable, and the repeatable items will be flagged with a dagger (†) character.

In the example below, option_1 and option_2 may be specified only once, but option_3 can be repeated multiple times.



—item_1—
—item_2—

A **Default Value** appears in underlined type. The example below shows a statement where YES is the default option; if no item is selected, the statement will perform exactly as if YES were typed in by the user:

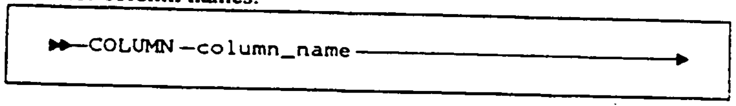


——KEYWORD——

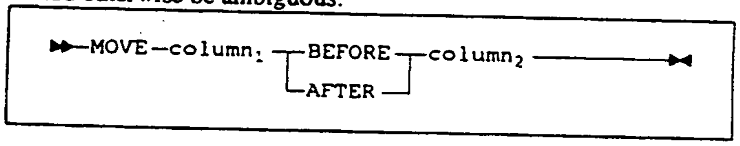
Keywords appear in all UPPERCASE letters, and must be entered exactly as shown (contractions or abbreviations are not allowed). If keywords are entered in lowercase, they will be folded to uppercase.

——variable——

Variables are shown in all lowercase letters, and indicate a point in the syntax where a substitution must be made by the user. In the following example, the variable column_name must be replaced with a 1- to 18-character-long identifier constructed according to DB2's rules for column names.

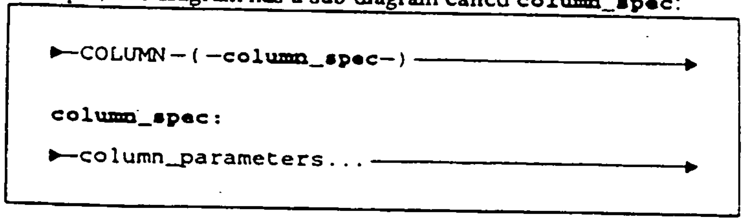


If a variable name occurs more than once, subscripted numbers are used to distinguish one occurrence from another when references would otherwise be ambiguous:



——**block**——
block:

Sub-Diagrams are indicated by variable names in **bold** type. When a portion of a syntax diagram cannot fit onto a single page, or within the boundary of a column, a variable name is used to represent a sub-diagram that is defined separately from the main diagram. For example, this diagram has a sub-diagram called **column_spec**:



Notice that, when the actual sub-diagram is defined, its name is followed by a colon, and does not appear on a syntax diagram line.

Variables Used in Syntax Diagrams

Note:
The following term definitions use the terms **short** and **long identifier** as defined in IBM'S *SQL Reference*, except that CDL does not currently support delimited identifiers. A **short identifier** is 8 characters; a **long identifier** is 18 characters.

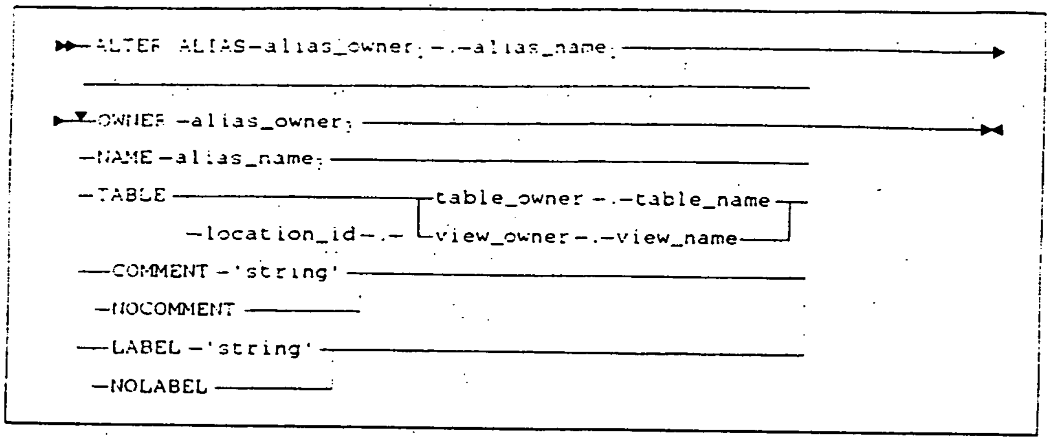
alias_name	A long identifier which names an alias.
alias_owner	A short identifier which names the owner of an alias.
auth_id	A short identifier recognized by the operating system as a valid authorization ID.
catalog_name	A short identifier that names a catalog.
column_name	A long identifier that names a column in a table or view.
constant	A value to be passed to a procedure; the type and syntax of a constant is dependent on the procedure, but will usually be a numeric or string value.
constraint_name	A short identifier that names a referential integrity constraint (foreign key definition).
database_name	A short identifier that names a database.
index_name	A long identifier that names an index.
index_owner	A short identifier that names the owner of an index.
location_id	A one- to 16-character-long identifier that designates a DB2 subsystem.
number	A DB2 INTEGER datum.
part_number	A number that identifies a partition in a partitioned index space or table space. When 0, identifies a non-partitioned index space or table space.

`password`	A short identifier recognized by the operating system as a valid password.
`stogroup_name`	A short identifier that names a storage group.
`subsystem_id`	A one- to four-character identifier that names a DB2 subsystem.
`synonym_name`	A long identifier that names a synonym.
`synonym_owner`	A short identifier that names the owner of a synonym.
`table_name`	A long identifier that names a table.
`table_owner`	A short identifier that names the owner of a table.
`tablespace_name`	A short identifier that names a tablespace.
`view_name`	A long identifier that names a view.
`view_owner`	A short identifier that names the owner of a view.
`volser`	A one- to six-character-long identifier that names a volume.

CDL Statements

This section presents each CDL statement individually. Each command description contains four parts: a syntax diagram, a brief description of the function of the statement, a description of each parameter used with the command, and an optional section on notes and warnings.

ALTER ALIAS



Description The ALTER ALIAS statement specifies changes to an alias definition.

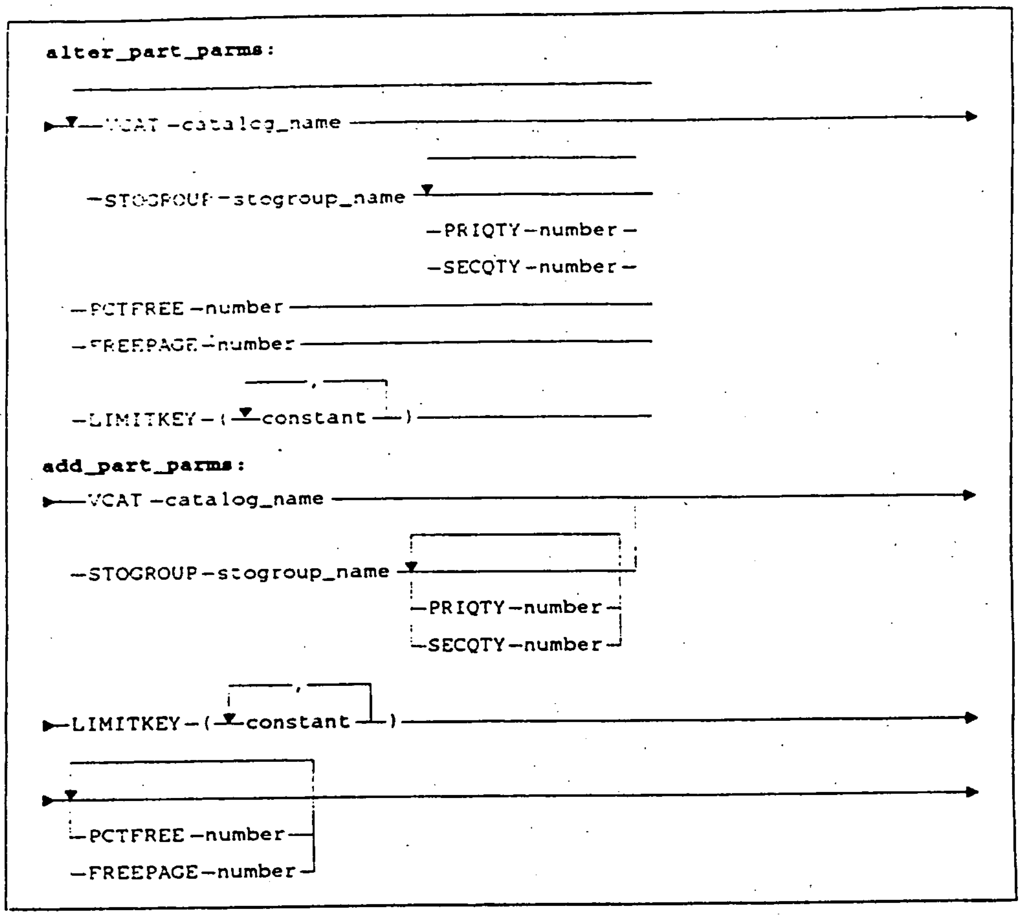
Parameters

alias_owner$_1$. alias_name$_1$
The name of the alias to be modified.

OWNER alias_owner$_2$
The new owner of the alias.

NAME alias_name$_2$
The new name of the alias.

TABLE (location_id .) *x*_owner . *x*_name
The new table or view referenced by the alias.



Parameters

`index_owner`$_1$. `index_name`$_1$
The name of the index to be changed.

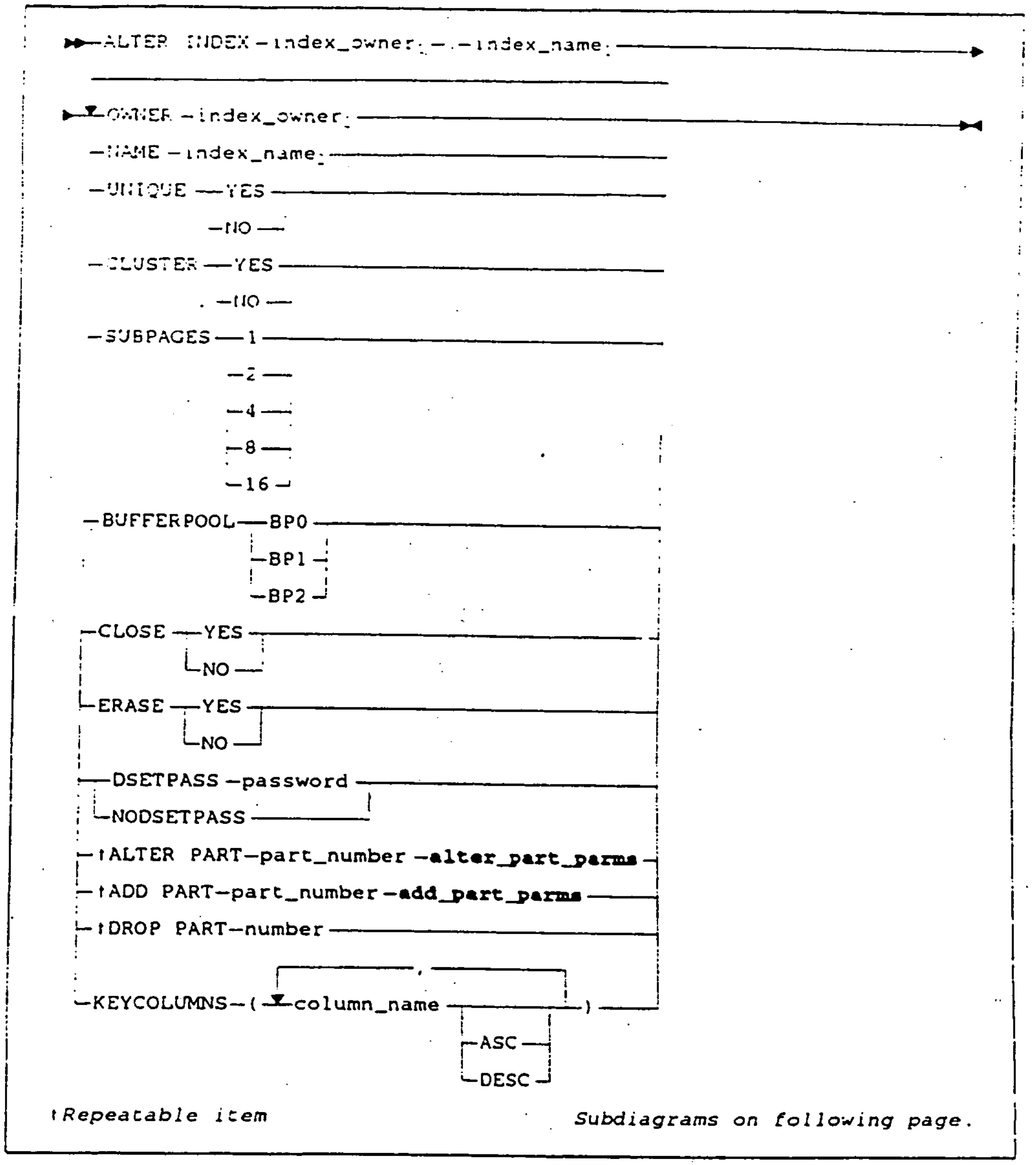
`OWNER index_owner`$_2$
The new owner of the index.

`NAME index_name`$_2$
The new name of the index.

`UNIQUE (YES / NO )`
Defines whether the index as unique or not.

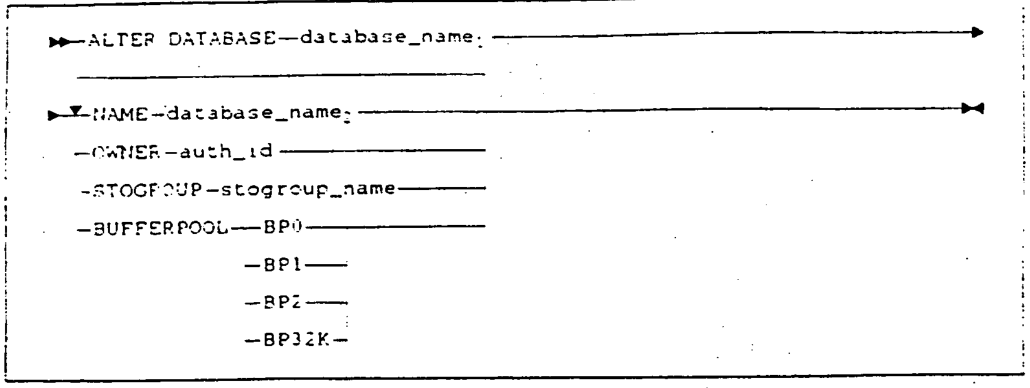
`CLUSTER (YES / NO)`
Defines the index as a clustering index.

ALTER INDEX



Description Description of this statement. The description should be a brief summary of the purpose, function, and use of the statement.

ALTER DATABASE



Description The ALTER DATABASE statement specifies changes to a database definition.

Parameters

database_name$_1$
The name of the database to be modified.

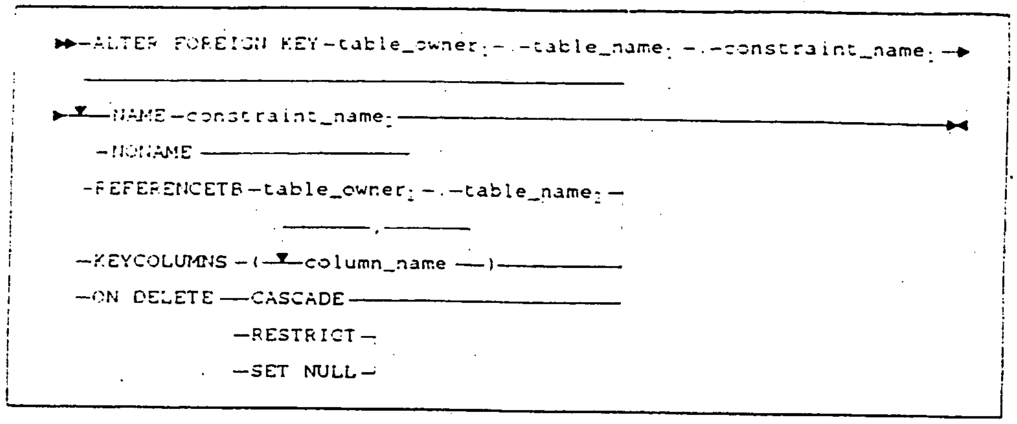
NAME database_name$_2$
The new name of the database.

OWNER auth_id
The new owner of the database.

STOGROUP stogroup_name
The new default storage group of the database.

BUFFERPOOL BPn
The new default bufferpool of the database.

ALTER FOREIGN KEY



Description The ALTER FOREIGN KEY statement defines changes to be made to a foreign key.

Parameters

table_owner1 . table_name1 . constraint_name1
The name of the foreign key to be changed.

NAME constraint_name2 / NONAME
The new name of the foreign key. If NONAME is specified, then the name of the foreign key will be generated by DB2.

REFERENCETB table_owner2 . table_name2
The new reference table of the foreign key.

KEYCOLUMNS (column_name_list)
The replacement columns to define the foreign key.

ON DELETE (CASCADE / RESTRICT / SET NULL)
The new delete rule for the foreign key.

SUBPAGES n
    The new number of subpages for the index.

BUFFERPOOL BPn
    The new bufferpool for the index.

CLOSE (YES / NO)
    The new index close parameter.

ERASE (YES / NO)
    The new erase rule for the index.

DSETPASS password / NODSETPASS
    The new data set password for the index. NODSETPASS removes a data set password.

ALTER PART part_number **alter_part_parms**
    For a partitioned index, alters part number according to the parameters in the block **alter_part_parms**. To alter parameters for a non-partitioned index, the part_number value must be zero (0). For example,

```
ALTER INDEX ...
ALTER PART 0 PCTFREE 20
```

ADD PART part_number **add_part_parms**
    For a partitioned index, adds a new part number according to the parameters in the block **add_part_parms**.

DROP PART number
    For a partitioned index, drops part number.

KEYCOLUMNS (column_name_list)
    The replacement column list that defines the key for this index.

VCAT catalog_name
    The new volume catalog to be used for the index.

STOGROUP stogroup_name
    The new default storage group for the index.

PRIQTY number
    The new primary allocation quantity.

SECQTY number
    The new secondary allocation quantity.

PCTFREE number
: The new quantity for percent of free space to retain in the index.

FREEPAGE number
: The new quantity for the number of free pages to retain in the index.

LIMITKEY (constant_list)
: The replacement key limit list for this part of a partitioned index.

ALTER STOGROUP

```
►►─ALTER STOGROUP─stogroup_name1─────────────────────────────►

►─▼─NAME─stogroup_name2──────────────────────────────────────►◄
   ─OWNER─auth_id──────
             ──,──
   ─VOLUMES─(─▼─volser──)─
   ─VCAT─catalog_name────
   ──VCATPASS─password──
    ─NOVCATPASS──────
```

Description The ALTER STOGROUP statement defines changes to be made to a storage group definition.

Parameters

`stogroup_name1`
The storage group to be changed.

`NAME stogroup_name2`
The new name of the storage group.

`OWNER auth_id`
The new owner of the storage group.

`VOLUMES (volser_list)`
The replacement list of volumes in the storage group.

`VCAT catalog_name`
The new catalog of the storage group.

`VCATPASS password / NOVCATPASS`
`VCATPASS password` defines a new password for the storage group, while NOVCATPASS defines the removal of a password.

ALTER SYNONYM

```
►►─ALTER SYNONYM─synonym_owner₁ ── .─synonym_name₁ ─────────────────►
  ──────────────────────────────────────────

►─▼─OWNER ─synonym_owner₂ ───────────────────────────────────────►◄
  ─NAME ─synonym_name₂ ───────────────────
  ─TABLE ─┬─table_owner ─.─table_name ─┬─
          └─view_owner ─.─view_name ───┘
```

Description The ALTER SYNONYM statement defines changes to be made to a DB2 synonym definition.

Parameters

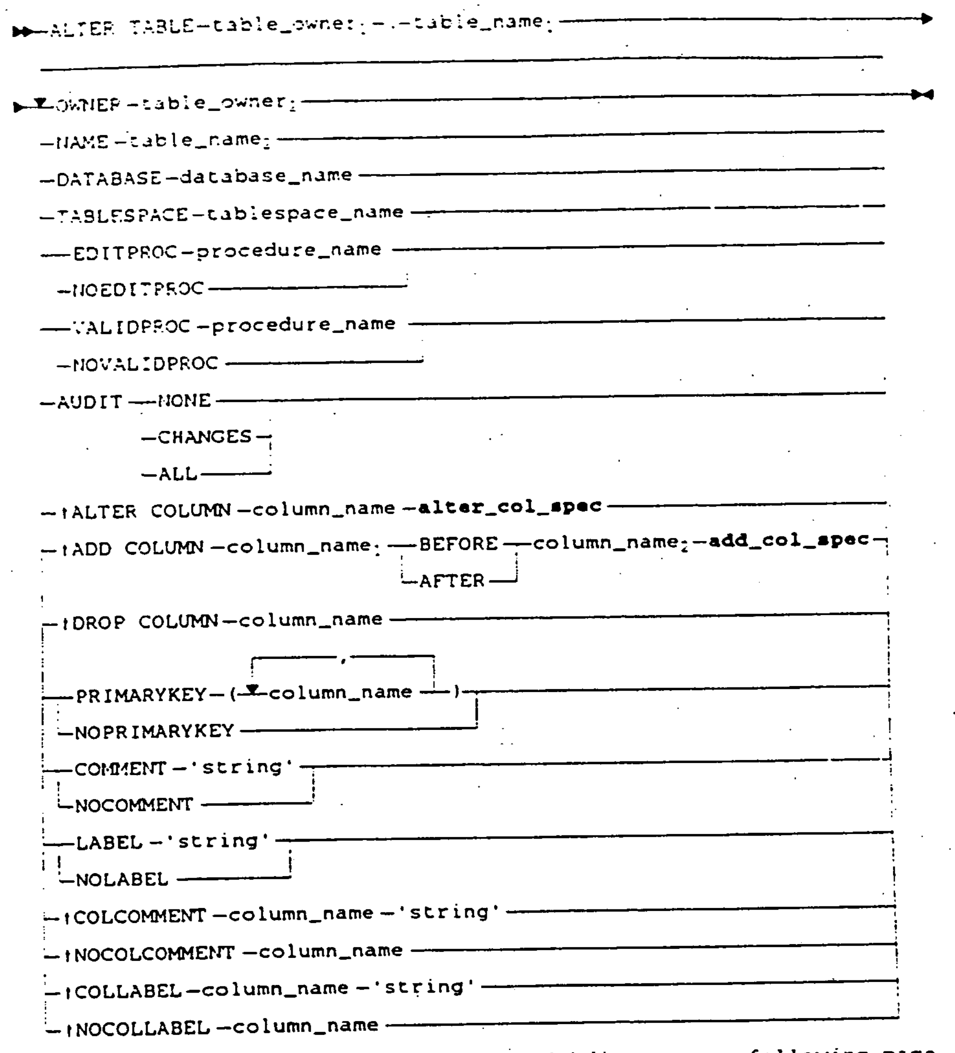
auth_id. synonym_name₁
: The name of the synonym to be changed.

OWNER synonym_owner₂
: The new owner of the synonym.

NAME synonym_name₂
: The new name of the synonym.

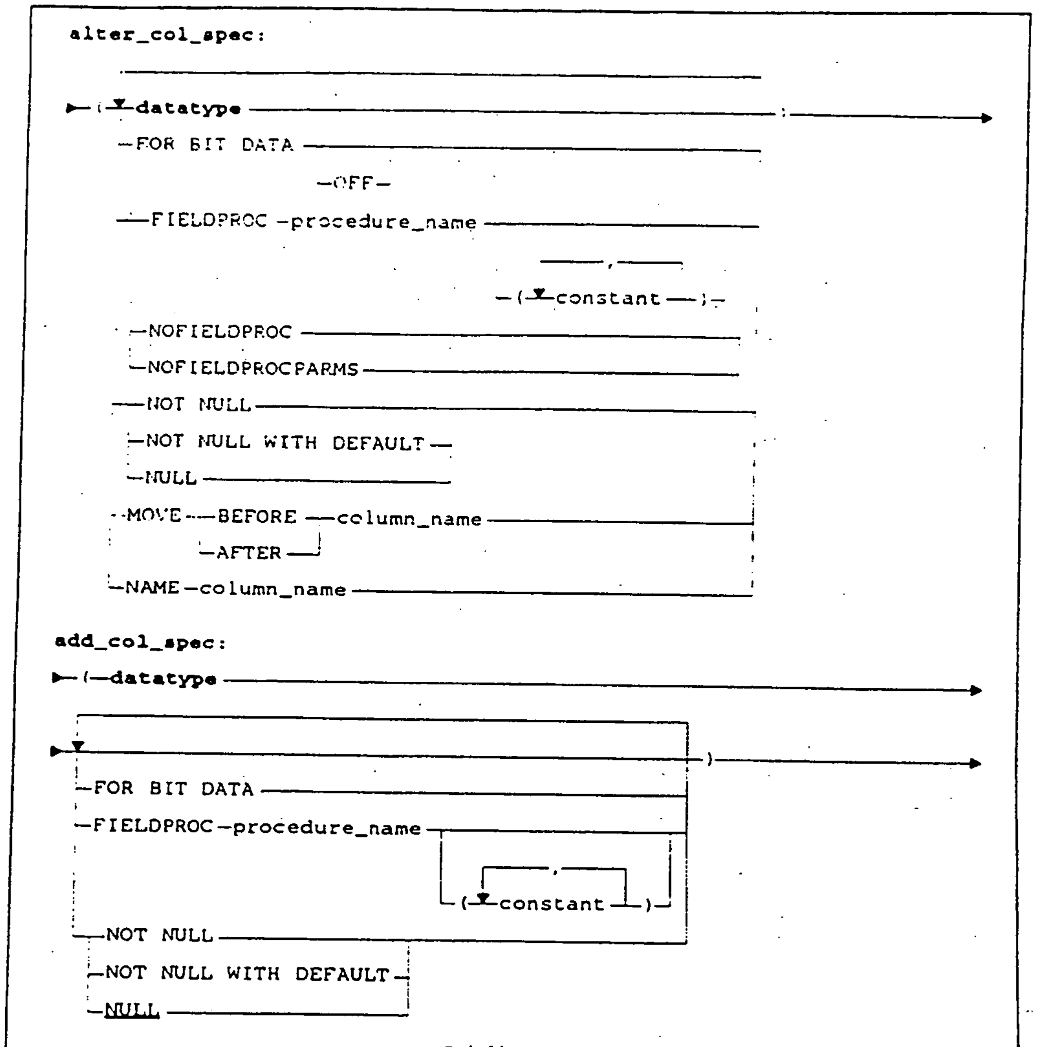
TABLE *x*_owner . *x*_name
: The new table or view referenced by the synonym.

ALTER TABLE


ALTER TABLE–table_owner.–.–table_name
OWNER–table_owner
NAME–table_name
DATABASE–database_name
TABLESPACE–tablespace_name
EDITPROC–procedure_name
NOEDITPROC
VALIDPROC–procedure_name
NOVALIDPROC
AUDIT–NONE
CHANGES
ALL
†ALTER COLUMN–column_name–**alter_col_spec**
†ADD COLUMN–column_name–BEFORE–column_name–**add_col_spec**
AFTER
†DROP COLUMN–column_name
PRIMARYKEY–(–column_name–)
,
NOPRIMARYKEY
COMMENT–'string'
NOCOMMENT
LABEL–'string'
NOLABEL
†COLCOMMENT–column_name–'string'
†NOCOLCOMMENT–column_name
†COLLABEL–column_name–'string'
†NOCOLLABEL–column_name

†Repeatable item *Subdiagrams on following page.*

Description The ALTER TABLE statement defines the changes to be made to a table.



Subdiagrams continued on following page.

Parameters

`table_owner`$_1$. `table_name`$_1$
The fully-qualified name of the table to be changed.

`OWNER table_owner`$_2$
The new owner of the table.

```
datatype:
▶──CHAR ─(─number ─)──────────────────────────────────────▶
  ─VARCHAR ─(─number ─)────────
  ─LONG VARCHAR ───────────────
  ─INTEGER ────────────────────
  ─SMALLINT ───────────────────
  ─FLOAT ─(─number ─)──────────
  ─DECIMAL ─(─number ──────────)─
                 ─,─number ─
  ─GRAPHIC ─(─number ─)────────
  ─VARGRAPHIC ─(─number ─)─────
  ─LONG VARGRAPHIC ────────────
  ─DATE ───────────────────────
  ─TIME ───────────────────────
  ─TIMESTAMP ──────────────────
```

NAME table_name$_2$
The new name of the table.

DATABASE database_name
The new database of the table.

TABLESPACE tablespace_name
The new tablespace of the table.

EDITPROC procedure_name / NOEDITPROC
EDITPROC defines a new edit procedure; NOEDITPROC removes an existing edit procedure.

VALIDPROC procedure_name / NOVALIDPROC
VALIDPROC defines a new validation procedure; NOVALIDPROC removes an existing validation procedure.

AUDIT (NONE / CHANGES / ALL)
The new audit procedure for the table.

ALTER COLUMN column_name **alter_col_spec**
Modifies column_name according to the specifications in the **alter_col_spec** block.

ADD COLUMN column_name$_1$ (BEFORE / AFTER) column_name$_2$ **add_col_spec**

Adds a new column column_name$_1$ before or after the existing column_name$_2$ according to the specifications in the **add_col_spec** block. column_name$_2$ must always be the current name–that is, the name of the column *before* any changes are applied by the current CDL statement–of an existing column in the table: if multiple columns are added or moved after the same column_name$_2$, they will be added or moved in the order specified. For example, the table X.TAB consists of three columns named COLA, COLB, and COLC. Executing this CDL command:

```
ALTER TABLE X.TAB
ALTER COLUMN COLB (NAME COLX)
ADD COLUMN COL1 AFTER COLB (INTEGER)
ADD COLUMN COL2 AFTER COLB (INTEGER)
```

would result in X.TAB having five columns (in order): COLA, COLX, COL1, COL2, and COLC. Note that, even though COLB was renamed COLX in the statement, COLB must be used in the ADD COLUMN clause, because the existing name of the column must be used.

However, this CDL statement:

```
ALTER TABLE X.TAB
ADD COLUMN COL1 AFTER COLB (INTEGER)
ADD COLUMN COL2 AFTER COL1 (INTEGER)
```

would result in an error since COL1 does not appear in the existing table X.TAB.

DROP COLUMN column_name

Removes the column column_name from the table.

PRIMARYKEY (column_name_list) / NOPRIMARYKEY

PRIMARYKEY creates a new primary key for the table consisting of the columns specified in the column name list. This primary key definition replaces an existing primary key or creates one if it does not exist. NOPRIMARYKEY removes a primary key definition from a table where one exists.

COMMENT 'string' / NOCOMMENT

COMMENT adds the comment in 'string' to the table; NOCOMMENT removes an existing comment definition.

LABEL 'string' / NOLABEL
LABEL adds the label in 'string' to the table; NOLABEL removes an existing label definition.

COLCOMMENT column_name 'string'
Adds the comment in 'string' to column_name.

NOCOLCOMMENT column_name
Removes the comment (if any) from column_name.

COLLABEL column_name 'string'
Adds the label in 'string' to column_name.

NOCOLLABEL column_name
Removes the label (if any) from column_name.

datatype
Defines the column to be of the specified DB2 data type.

Note:
FLOAT(number) and DECIMAL(number[,number) ...

FOR BIT DATA {OFF}
Defines the column to be for bit data.

FIELDPROC procedure_name ((constant_list)) / NOFIELDPROC
FIELDPROC defines a field procedure for the column; NOFIELDPROC removes a field procedure definition.

NOT NULL / NOT NULL WITH DEFAULT / NULL
Defines the null handling attributes of the field.

MOVE {BEFORE / AFTER} column_name
Moves the column before or after column_name (which must be the current name of an existing column). For example, if the table X.TAB contains the columns COL1, COL2, and COL3, then this CDL statement:

```
ALTER TABLE X.TAB
ALTER COLUMN COL1 (NAME COLX)
ALTER COLUMN COL2 (MOVE BEFORE COL1)
```

would result in a column order of COL2-COLX-COL3. However, this CDL statement

```
ALTER TABLE X.TAB
ADD COLUMN COLA BEFORE COL1 (col_spec...
ALTER COLUMN COL2 (MOVE BEFORE COLA)
```

would result in an error, since COLA does not exist in the table before the invocation of the CDL statement.

NAME column_name
The new name of the column.

Notes and Warnings

The ADD COLUMN and ALTER COLUMN clauses are positional in application. That is, if the CDL specifies these three clauses

```
ALTER TABLE ...
ADD COLUMN COL1 ...
ADD COLUMM COL2 ...
ADD COLUMN COL3 ...
```

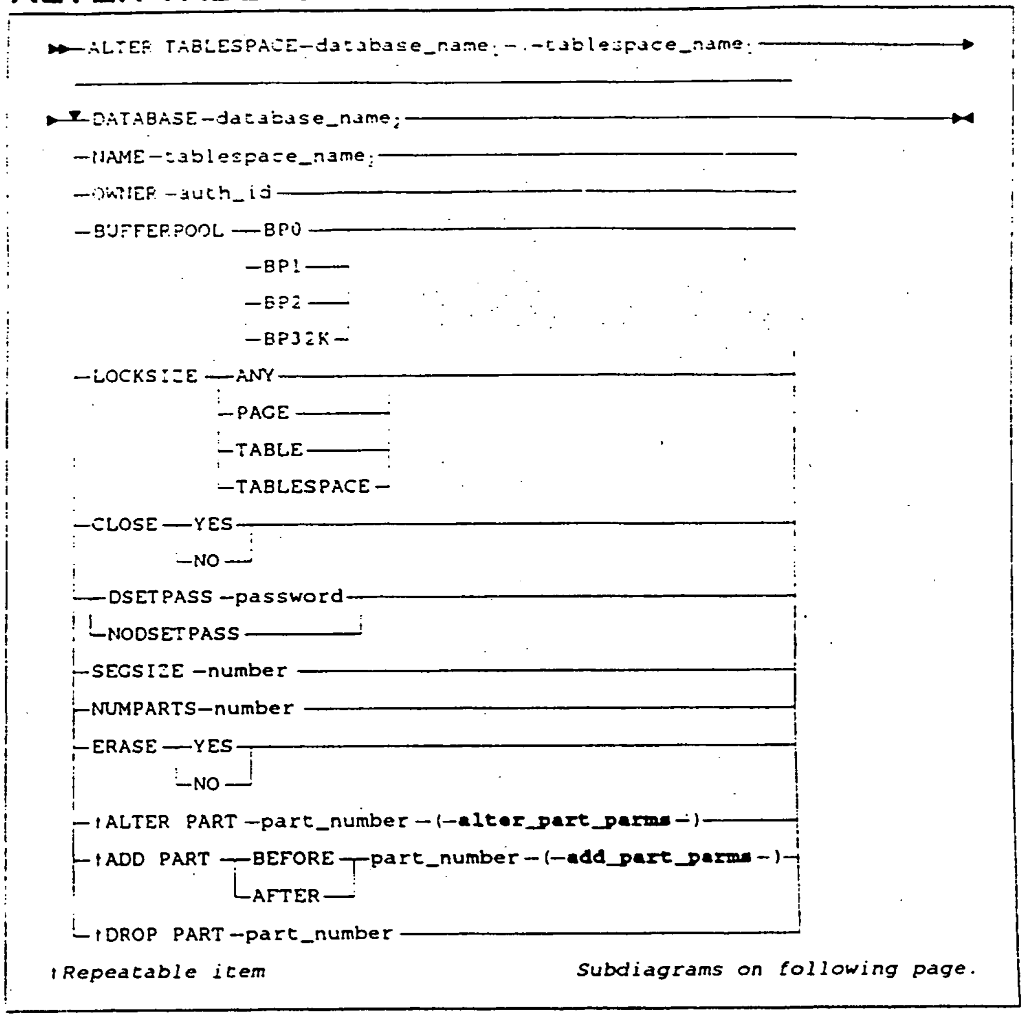
The columns will be added in the order specified, and will appear in that order in the table.

For a further example, if table X.TAB contains COL1, COL2, and COL3, this CDL statement:

```
ALTER TABLE X.TAB
ADD COLUMN COL4NEW AFTER COL1 (FLOAT(21))
ALTER COLUMN COL3 (MOVE AFTER COL1)
```

would result in a table order of COL1-COL4NEW-COL3-COL2.

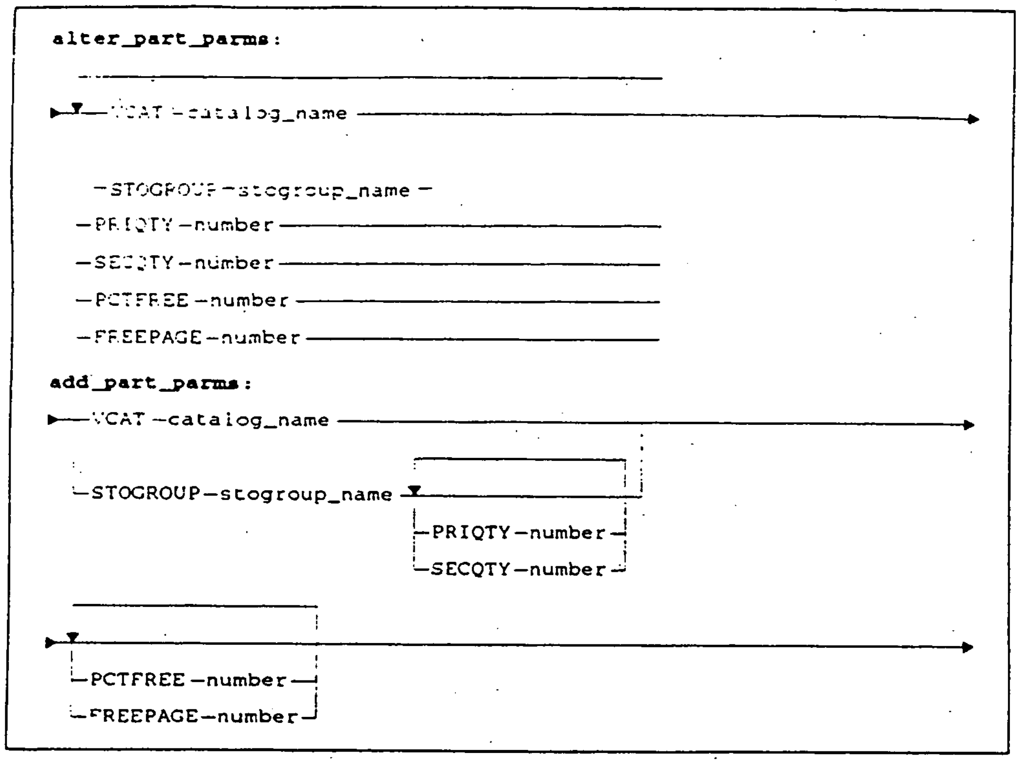
ALTER TABLESPACE



Description The ALTER TABLESPACE statement specifies the changes necessary to make a remote tablespace identical to a local tablespace.

Parameters `DATABASE database_name₂`
The new database for this tablespace.

`NAME tablespace_name₂`
The new name for this tablespace.



OWNER auth_id
The new owner of the tablespace.

ALTER PART part_number (**alter_part_parms**)
For a partitioned tablespace (NUMPARTS > 1), modifies part part_number according to the parameters in the **alter_part_parms** block.

For a non-partitioned table space, part_number must be zero (0).

ADD PART part_number (**add_part_parms**)
For a partitioned tablespace, adds a new part part_number according to the parameters in the **add_part_parms** block.

DROP PART part_number
For a partitioned tablespace, removes the definition of part part_number.

BUFFERPOOL BPn
: The new default bufferpool for the tablespace.

LOCKSIZE
: The new lock size for the tablespace.

CLOSE (YES / NO)
: The new close parameter of the tablespace.

DSETPASS password / NODSETPASS
: DSETPASS password defines a new data set password for the tablespace, while NODSETPASS removes a data set password definition.

SEGSIZE number
: The new segment size of the table space, or 0 to specify a non-segmented tablespace. DB2 requires SEGSIZE to be between 4 and 64 and divisible by four if segmented.

NUMPARTS number
: The new number of parts for the table space. NUMPARTS=0 specifies a non-partitioned table space.

ERASE (YES / NO)
: The new erase parameter of the tablespace.

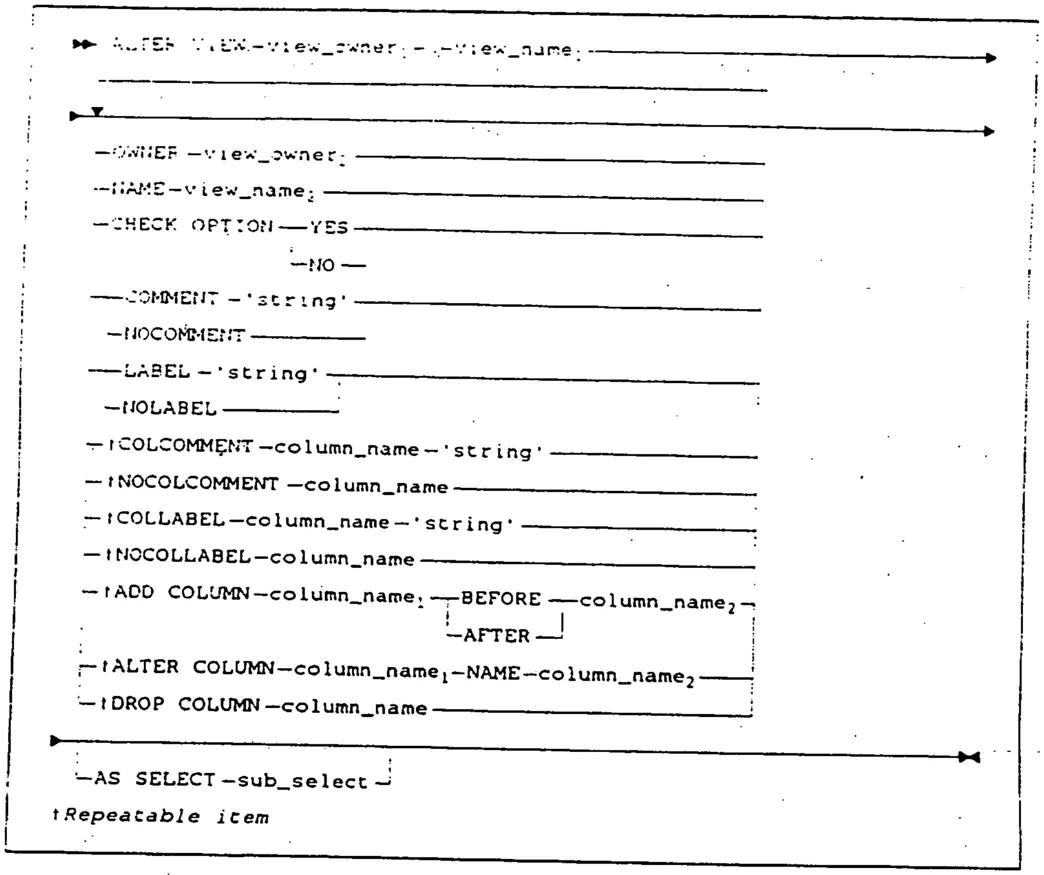
PRIQTY number
: The new primary allocation quantity (valid only with STOGROUP, not VCAT).

SECQTY number
: The new secondary allocation quantity (valid only with STOGROUP, not VCAT).

PCTFREE number
: The percent of free space to be left on each page of the tablespace upon create or reorganization.

FREEPAGE number
: Defines that a free page will be added after every number pages upon tablespace creation or reorganization.

ALTER VIEW


ALTER VIEW–view_owner₁ . –view_name₁
–OWNER –view_owner₂
–NAME–view_name₂
–CHECK OPTION –YES
–NO
–COMMENT –'string'
–NOCOMMENT
–LABEL –'string'
–NOLABEL
–†COLCOMMENT –column_name –'string'
–†NOCOLCOMMENT –column_name
–†COLLABEL–column_name –'string'
–†NOCOLLABEL–column_name
–†ADD COLUMN–column_name₁ –BEFORE –column_name₂
–AFTER
–†ALTER COLUMN–column_name₁–NAME–column_name₂
–†DROP COLUMN–column_name
–AS SELECT–sub_select
†Repeatable item

Description The ALTER VIEW statement defines changes to be made to an existing view definition.

Parameters view_owner$_1$. view_name$_1$

The fully-qualified name of the view to be modified.

OWNER view_owner$_2$

The new owner of the view.

NAME view_name$_2$

The new name of the view.

CHECK OPTION (YES / NO)
: The new check option for the view.

COMMENT 'string' / NOCOMMENT
: COMMENT 'string' defines a new comment for the view, while NOCOMMENT removes an existing comment.

LABEL 'string' / NOLABEL
: LABEL 'string' defines a new label for the view, while NOLABEL removes an existing label.

COLCOMMENT column_name 'string'
: Defines a comment 'string' on column_name.

NOCOLCOMMENT column_name
: Removes an existing column comment from column_name.

COLLABEL column_name 'string'
: Defines a label 'string' on column_name.

NOCOLLABEL column_name
: Removes an existing column label from column_name.

ADD COLUMN column_name$_1$ (BEFORE / AFTER) column_name$_2$
: Adds a new column column_name$_1$ to the view before or after the existing column column_name$_2$ (which must be the *current* name of the column).

ALTER COLUMN column_name$_1$ NAME column_name$_2$
: Modifies the column column_name$_1$ giving it the new name column_name$_2$.

DROP COLUMN column_name
: Drops column_name from the view definition.

AS SELECT sub_select
: Defines a new SQL subselect statement as the definition for the view. If specified, the new select statement must appear last in the ALTER VIEW statement, after any other clauses.

Note:

CDL does not support the WITH CHECK OPTION clause of DDL. Use CHECK OPTION (YES/NO) instead. SEE Create View.

CREATE ALIAS

```
▶▶─CREATE ALIAS─alias_owner─.─alias_name──────────────────────▶

▶─TABLE─────────────────┬─table_owner─.─table_name─┬──────────▶
       ─location_id─.─  └─view_owner─.─view_name ──┘
   ──────────────────────

▶─▼───────────────────────────────────────────────────────────▶◀
  ─COMMENT─'string'─
  ─LABEL─'string'───
```

Description The CREATE ALIAS statement creates a new alias for a remote table.

Parameters

`alias_owner . alias_name`
The name of the alias to be created.

`TABLE (location_id .) table_owner . table_name`
The qualified name of the table or view referenced by the alias.

`COMMENT 'string'`
An optional comment for the alias.

`LABEL 'string'`
An optional label for the alias.

CREATE DATABASE

```
▶▶—CREATE DATABASE—database_name————————————————————————————▶
   ———————————————————————————————
▶—▼——————————————————————————————————————————————————————————▶◀
   —OWNER—auth_id ————————————————
   —STOGROUP—stogroup_name————————
   —BUFFERPOOL——BP0———————————————
                —BP1———
                —BP2———
                —BP32K—
```

Description The CREATE DATABASE statement creates a new database.

Parameters

OWNER auth_id
: The owner of the database.

STOGROUP stogroup_name
: Assigns this database to the new default storage group stogroup_name. If the STOGROUP keyword is not specified, the default **SYSDFLT** storage group will be used.

BUFFERPOOL BPn
: The default buffer pool to be associated with the database.

CREATE FOREIGN KEY

```
▶▶─CREATE FOREIGN KEY─table_owner1─.─table_name1───────────────────▶
                                                  └─.─constraint_name─┘
▶─REFERENCETB ─table_owner2─.─table_name2 ─────────────────────────▶

                  ──────── , ────────
▶─KEYCOLUMNS ─(─▼─column_name ──)──────────────────────────────────▶
▶──────────────────────────────────────────────────────────────────▶◀
   └─ON DELETE ──CASCADE───┘
                 ─RESTRICT─
                 ─SET NULL─
```

Description The CREATE FOREIGN KEY statement defines a foreign key for a table's referential integrity checking.

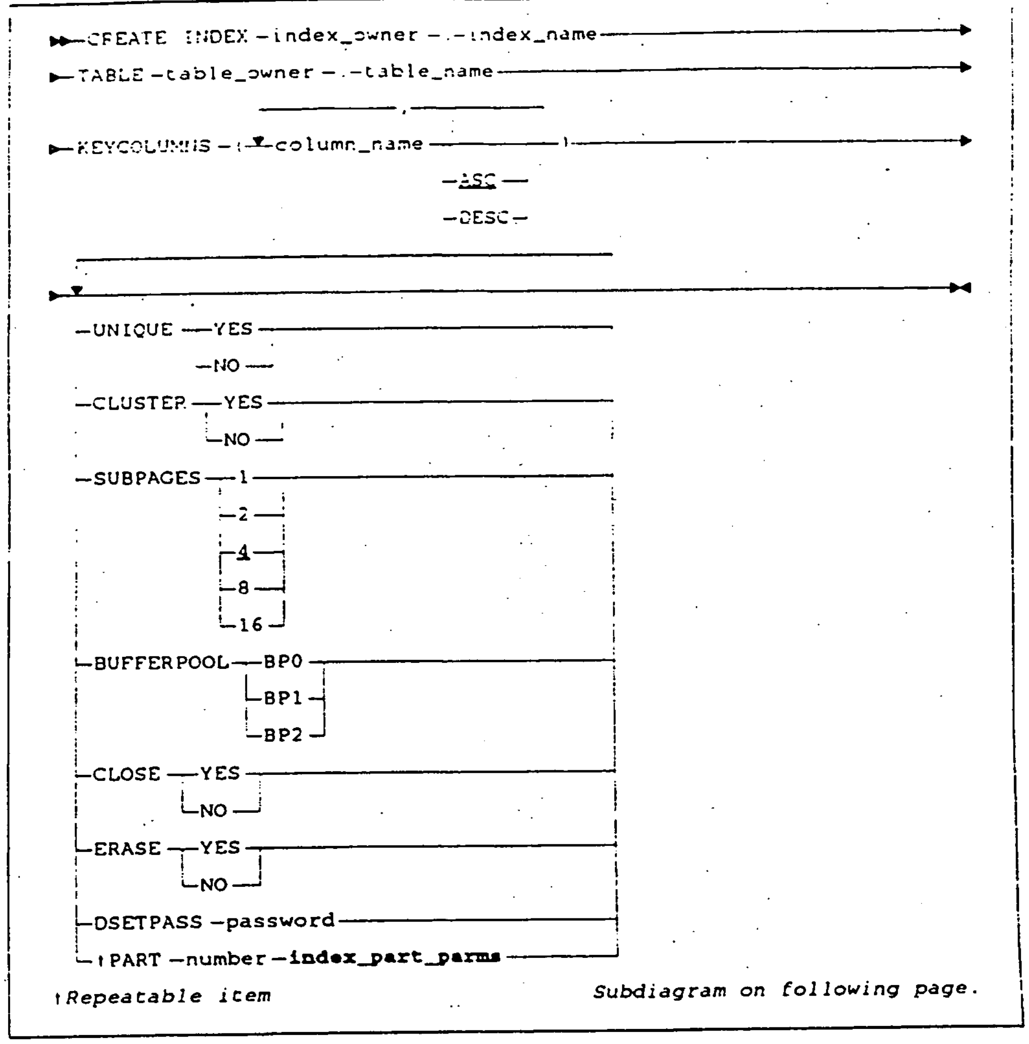
table_owner$_1$. table_name$_1$ (. constraint_name)
: The qualified constraint-name of the foreign key. If constraint_name is not specified, a name will be generated automatically by DB2.

REFERENCETB table_owner$_2$. table_name$_2$
: The table referenced by the foreign key.

KEYCOLUMNS (column_name_list)
: The columns to be included in the foreign key definition.

ON DELETE (CASCADE / <u>RESTRICT</u> / SET NULL)
: The delete behavior of the referential integrity constraint.

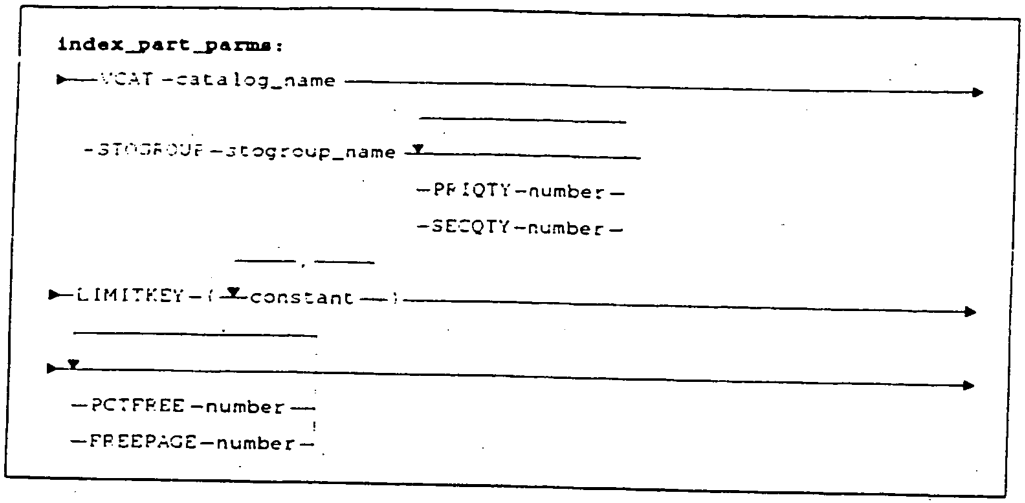
CREATE INDEX



Description The CREATE INDEX statement defines a new index for a table.

index_owner . index_name
The owner and name of the new index.

CREATE INDEX



`TABLE table_owner . table_name`
The table the index is being created on.

`KEYCOLUMNS (column_name_list (ASC / DESC))`
The names of the columns to be included in the index, along with a keyword specifying the sort order (ASCending or DESCending).

`UNIQUE (YES / NO)`
Defines whether or not index values must be unique.

`CLUSTER (YES / NO)`
Defines this index as a clustering index, if YES is specified.

`SUBPAGES (1 / 2 / 4 / 8 / 16)`
Defines the number of subpages for the index.

`BUFFERPOOL BPn`
Defines the default buffer pool to be used by the index.

`CLOSE (YES / NO)`
Defines the close action of the index.

`ERASE (YES / NO)`
Defines the erase action of the index.

`DSETPASS password`
Defines a data set password for the index.

PART number **index_part_parms**
For a partitioned index, defines the parameters for part number according to the block **index_part_parms**.

VCAT catalog_name
The catalog to be used for the index.

STOGROUP stogroup_name
The default storage group for the index.

PRIQTY number
The primary allocation quantity for the index.

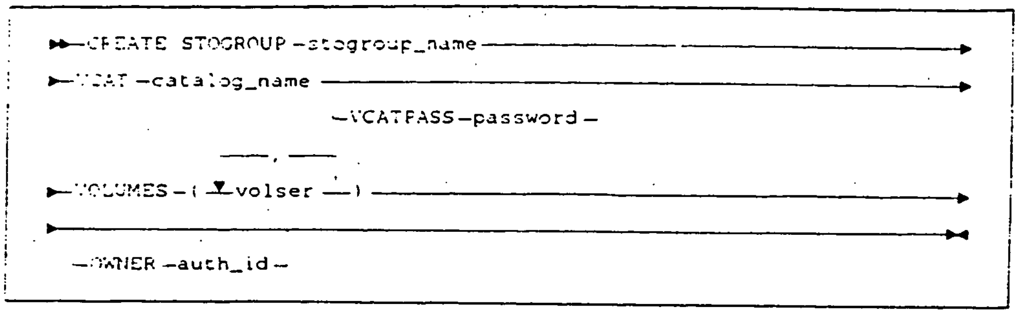
SECQTY number
The secondary allocation quantity for the index.

LIMITKEY (constant_list)
The limiting constants for this part of the index.

PCTFREE number
The percentage of free space to leave in the index space upon create or reorganization.

FREEPAGE number
Defines a free page every number pages in the index.

CREATE STOGROUP



Description The CREATE STOGROUP statement defines a new storage group to DB2.

Parameters

`stogroup_name`
: The name of the storage group.

`VCAT catalog_name`
: The catalog of the storage group.

`VOLUMES (volser_list)`
: A list of volumes to be included in the storage group.

`OWNER auth_id`
: The owner of the storage group.

`VCATPASS password`
: Defines a password to be used for the volume catalog.

CREATE SYNONYM

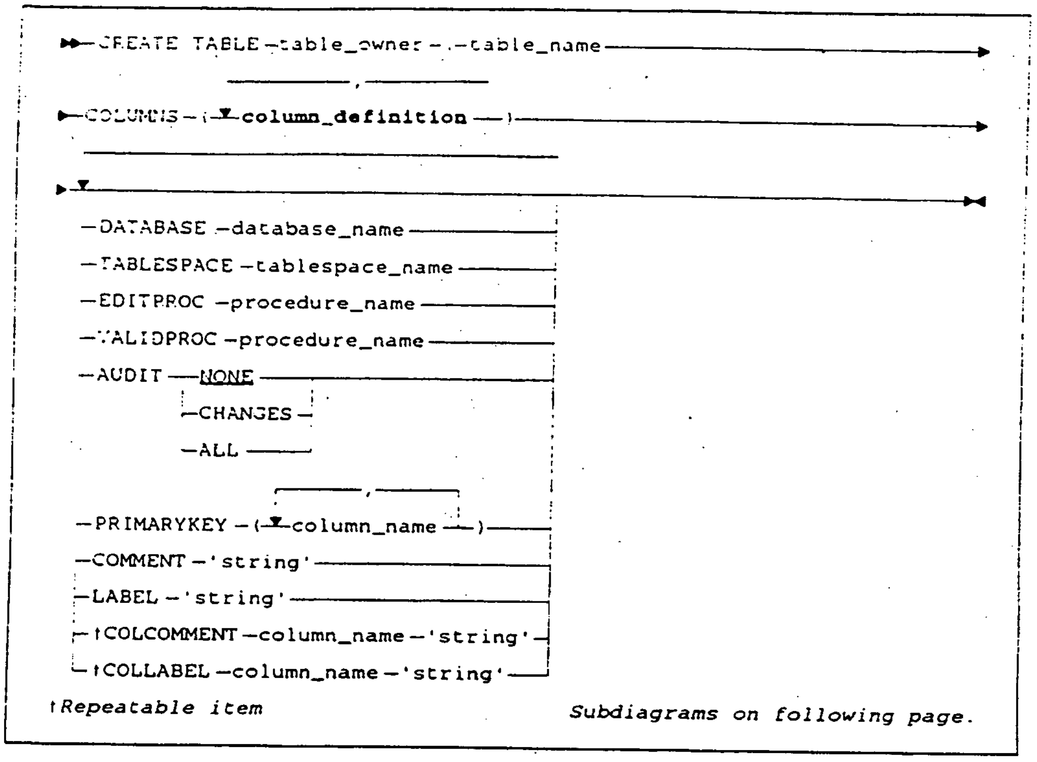
```
▶▶─CREATE SYNONYM─synonym_owner─.─synonym_name──────────────────────▶

▶─TABLE─┬─table_owner─.─table_name─┬─────────────────────────────────▶◀
        └─view_owner─.─view_name───┘
```

Description The CREATE SYNONYM statement creates a synonym for an existing table.

Parameters `synonym_owner . synonym_name`
The owner and name of the synonym to be created.

`TABLE x_owner . x_name`
The table or view referenced by the synonym.

CREATE TABLE



Description The CREATE TABLE statement defines a new table to the DB2 system.

Parameters

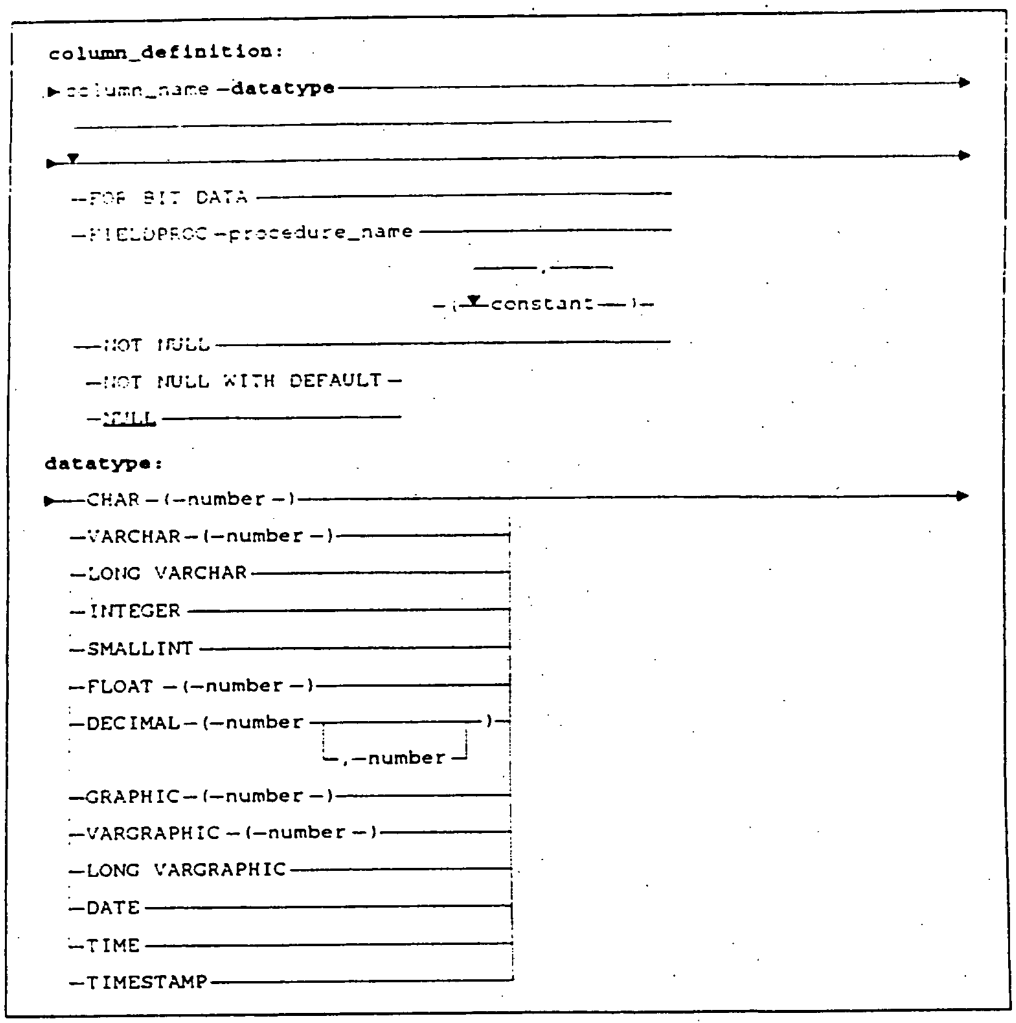
`table_owner . table_name`
The owner and name of the table to be created.

COLUMNS (**column_definition_list**)
The column definitions for the table according to the **column_definition** block.

DATABASE database_name
The database for the table.

TABLESPACE tablespace_name
The tablespace used by the table.

EDITPROC procedure_name
An edit procedure for the table.



`VALIDPROC procedure_name`
A validation procedure for the table.

`AUDIT (NONE / CHANGES / ALL)`
The audit activity for the table.

`PRIMARYKEY (column_name_list)`
The primary key definition for the table.

CREATE.TABLE

COMMENT 'string'
A comment for the table.

LABEL 'string'
A label for the table.

COLCOMMENT column_name 'string'
A column comment on column_name.

COLLABEL column_name 'string'
A column label on column_name.

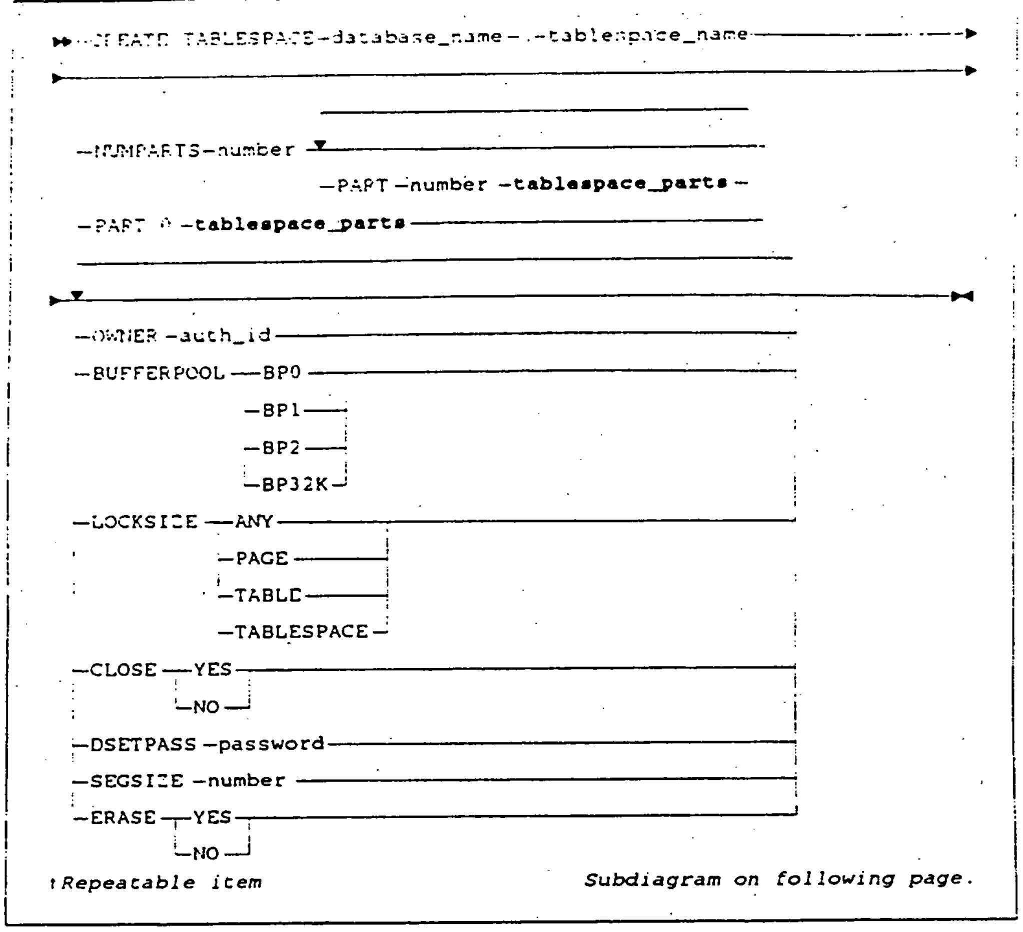
datatype
The DB2 data type for this column.

FOR BIT DATA
For character columns, defines that the column should be treated as binary data, not EBCDIC characters.

FIELDPROC procedure_name ((constant_list))
Defines a field procedure for the defined column.

NOT NULL / NOT NULL WITH DEFAULT / NULL
Defines the null activity for the column.

CREATE TABLESPACE

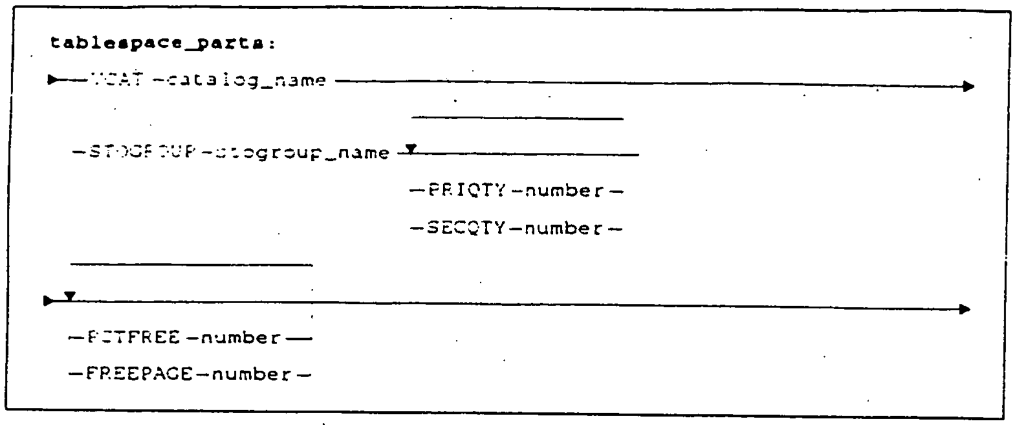


```
▶▶—CREATE TABLESPACE—database_name—.—tablespace_name——————————▶

▶———————————————————————————————————————————————————————————————▶

    —NUMPARTS—number —▼——————————————————————————
                        —PART—number —tablespace_parts—
    —PART 0 —tablespace_parts —————————————————————

▶—▼—————————————————————————————————————————————————————————————▶◀
    —OWNER —auth_id ————————————————————————————
    —BUFFERPOOL ——BP0 ——————————————————————————
                  —BP1——
                  —BP2——
                  —BP32K—
    —LOCKSIZE ——ANY ————————————————————————————
                —PAGE ———
                —TABLE———
                —TABLESPACE—
    —CLOSE ——YES————————————————————————————————
             —NO——
    —DSETPASS —password ————————————————————————
    —SEGSIZE —number ———————————————————————————
    —ERASE ——YES————————————————————————————————
             —NO——
†Repeatable item                    Subdiagram on following page.
```

Description The CREATE TABLESPACE command defines a new tablespace to a DB2 system.

Parameters `database_name . tablespace_name`
The database and name of the tablespace to be created.

CREATE TABLESPACE



NUMPARTS number (PART number tablespace_parts) / PART 0 tablespace_parts
: If NUMPARTS > 0, then a partitioned tablespace is being created, and each part must be specified according to the part number and the parameters in the **tablespace_parts** block. If PART 0 is specified, then the tablespace is non-partitioned and the **tablespace_parts** block is used to define it.

OWNER auth_id
: The owner of the tablespace.

BUFFERPOOL BPn
: The default bufferpool associated with this tablespace.

LOCKSIZE (ANY / PAGE / TABLE / TABLESPACE)
: The lock size parameter for the tablespace.

CLOSE (YES / NO)
: The close action specification of the tablespace.

DSETPASS password
: A data set password to be used for the tablespace.

SEGSIZE number
: The segment size of the tablespace.

ERASE (YES / NO)
: The erase action of the tablespace.

VCAT catalog_name
: The volume catalog to be used by the tablespace.

STOGROUP stogroup_name
The default storage group for the tablespace.

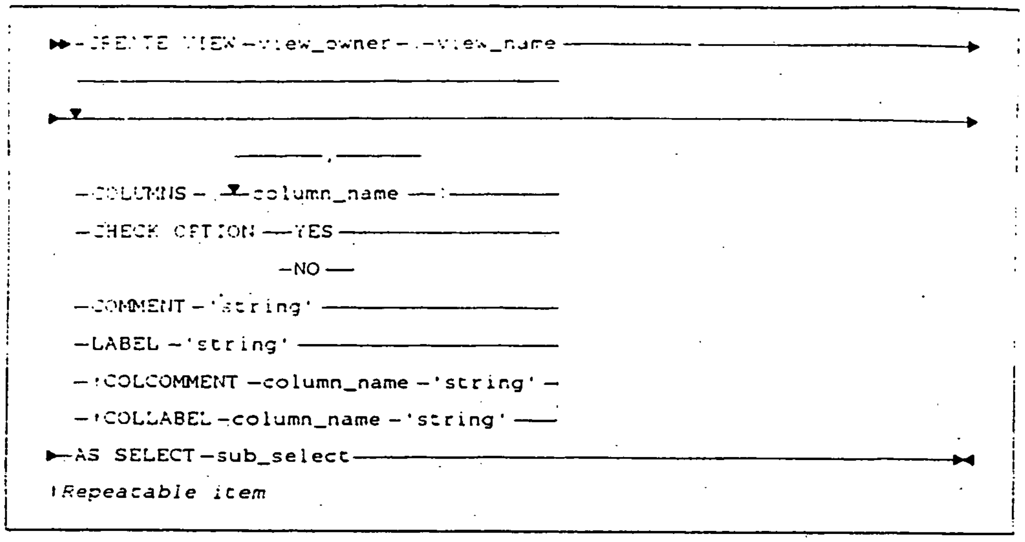
PRIQTY number
The primary allocation quantity of the tablespace.

SECQTY number
The secondary allocation quantity of the tablespace.

PCTFREE number
The percent of free space to be left on each page of the tablespace upon create or reorganization.

FREEPAGE number
Defines that a free page will be added after every number pages upon tablespace creation or reorganization.

CREATE VIEW



Description The CREATE VIEW statement defines a new view.

Parameters

`view_owner . view_name`
The name and owner of the view being created.

`COLUMNS (column_name_list)`
The names of the columns to be included in the view.

`CHECK OPTION (YES / NO)`
The check option to be used by the view.

`COMMENT 'string'`
The comment on the view.

`LABEL 'string'`
The label for the view.

`COLCOMMENT column_name 'string'`
Defines a comment on `column_name`.

`COLLABEL column_name 'string'`
Defines a label for `column_name`.

`AS SELECT sub_select`
Defines the SQL subselect statement used to create the view. The `SELECT` clause is required, and it must appear last in the `CREATE VIEW` statement.

Note:
CDL does not support the `WITH CHECK OPTION` clause of SQL. Use `CHECK OPTION (YES/NO)` instead.

DROP ALIAS

```
►►—DROP ALIAS —alias_owner — . —alias_name————————————————►◄
```

Description The DROP ALIAS statement removes an alias definition.

Parameters `alias_owner . alias_name`
The owner and name of the alias to be dropped.

DROP DATABASE

```
►►─DROP DATABASE─database_name──────────────────────────►◄
```

Description The DROP DATABASE statement removes a database and all dependent objects.

Parameters `database_name`
The name of the database being dropped.

DROP FOREIGN KEY

```
▶▶—DROP FOREIGN KEY—table_owner—.—table_name—.—constraint_name———▶◀
```

Description The DROP FOREIGN KEY statement removes a referential integrity constraint from a table.

Parameters `table_owner . table_name . constraint_name`
The fully-qualified name of the constraint to be dropped.

DROP INDEX

```
▶▶—DROP INDEX—index_owner—.—index_name————————————————————▶◀
```

Description The DROP INDEX statement removes an index from a table.

Parameters `index_owner . index_name`
The name of the index being dropped.

DROP STOGROUP

```
►►—DROP STOGROUP—stogroup_name——————————————————————————————►◄
```

Description The DROP STOGROUP statement removes a storage group definition.

Parameters `stogroup_name`
The name of the storage group being dropped.

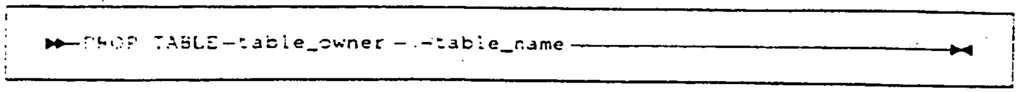
DROP SYNONYM

```
►►─DROP SYNONYM ─synonym_owner ─ . ─synonym_name ──────────────►◄
```

Description The DROP SYNONYM statement removes a synonym definition.

Parameters synonym_owner . synonym_name
The name of the synonym being dropped.

DROP TABLE



Description The DROP TABLE statement removes a table definition.

Parameters `table_owner . table_name`
The name of the table being dropped.

DROP TABLESPACE

```
▶▶—DROP TABLESPACE —database_name—.—tablespace_name————————▶◀
```

Description The DROP TABLESPACE statement removes a tablespace and all dependent objects.

Parameters `database_name . tablespace_name`
The fully-qualified name of the tablespace being dropped.

DROP VIEW

```
▸▸—DROP VIEW—view_owner—.—view_name——————————————————▸◂
```

Description The DROP VIEW removes a view definition.

Parameters view_owner . view_name

The fully-qualified name of the view being dropped.

Dash Commands

CDL File Format

CDL is imported into CHANGE MANAGEMENT from sequential, 80-column files. Each command begins on a separate line, though commands may be continued on multiple lines. Command record formats are shown below.

First Line of Commands

COLUMN(S)	USAGE
1	Must be dash "-"
2-5	Command name (e.g., DEST)
6	Must be blank
7-12	Sequence number (zero-filled)
13	Must be blank
14-72	Free format command text
73-80	Reserved

Command Continuation Lines

COLUMN(S)	USAGE
1	Must be blank
2-72	Free format command text
73-80	Reserved

There is no implied space or other token separator between column 72 of one line and column 2 of the next. For example, if "TABLE" appears in columns 68-72 of one line, and "SPACE" appears in columns 2-6 of the next, the CDL scanner will read the token "TABLESPACE", and not the two words "TABLE" and "SPACE". Quoted strings can be continued from one line to the next.

Command Termination

Commands are terminated upon reaching either the end of the file or the first following line which has a non-blank character in column one.

Command Sequence

The first three commands in a CDL file must be (in order) -TIME, -ORGN, and -DEST.

Comments

Comments are signalled by an asterisk "*" in column 1, and terminate at the end of the line (column 80). Comments may appear anywhere in a file except in the middle of a command (since commands are terminated by a non-blank character in column one).

-CDL: Change Definition Language Statement

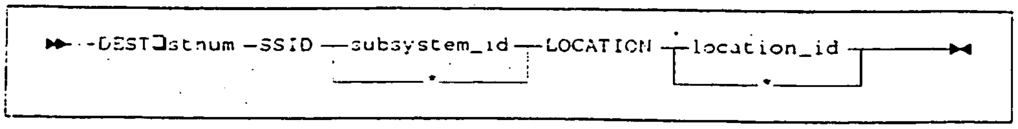
```
▶▶—-CDL□□stnum -cdl_statement ——————————————————————————▶◀
```

Description The -CDL command is used to introduce a CDL statement.

Parameters `-CDL□□stnum`
The □□ symbol indicates that the -CDL token must be followed by two spaces, since `stnum` must appear in columns 7-12.

`cdl_statement`
A valid CDL statement as defined in the preceding section.

-DEST: Destination

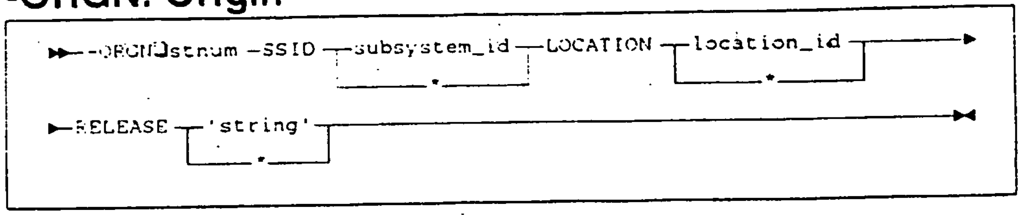


Description The -DEST command defines the intended destination system of the CDL file. -DEST must be the third command in a CDL file (after -TIME and -ORGN).

Parameters `SSID subsystem_id`
The destination subsystem of the CDL file. May be '*' if unknown.

`LOCATION location_id`
The destination location of the CDL file. May be '*' if unknown.

-ORGN: Origin



Description The -ORGN statement defines the system that originated the CDL. -ORGN is a required command; it must appear after -TIME and before -DEST.

Parameters `SSID subsystem_id`
The origin subsystem of the CDL statements. May be '*' if unknown.

`LOCATION location_id`
The origin location ID of the CDL. May be '*' if unknown.

`RELEASE 'string'`
The version and release number of the DB2 system from which the CDL was generated. This is the release string returned from a DB2 CALL ATTACH.

-TIME: Creation Time

```
▶▶—-TIME—strnum—'timestamp_string'——————————————————▶◀
```

Description The -TIME statement defines the time and date the CDL file was created. -TIME must be the first executable command in the CDL file.

Parameters 'timestamp_string'
A 26-byte string, in DB2 TIMESTAMP format, that defines the date and time the CDL file was created.

APPENDIX B

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
%{

#include "acmp.h"
#ifdef ACMG_OS2
#include <memory.h>
#endif
#ifdef ACMG_MVS
#include <string.h>
#endif


extern struct ACMP_P acmpdata;
extern SHORT acmplineno;
extern SHORT acmp_retcode;
extern SHORT acmp_reason;       /* REASON CODE FOR ERROR */
extern BOOL acmp_semerror;

%}
%prefix acmpc
%union {
 struct ACMP_ID strval;
 VCHAR *idval;
 short numval;
 char byteval;
 struct ACMP_TN *tbnam;
}


%type <idval> stogroupparm
%type <idval> shortid
%type <idval> ownerparm
%type <idval> newsname
%type <idval> newlname
%type <idval> vcatpass
%type <idval> volser
%type <idval> dbasename
%type <idval> passwparm
%type <idval> longid
%type <idval> locationid
%type <idval> tgttype
%type <numval> freepageparm
%type <numval> pctfreeparm
%type <numval> priqtyparm
%type <numval> secqtyparm
%type <numval> partsparm
%type <numval> segparm
%type <byteval> unit
%type <byteval> eraserule
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
%type <byteval> yesorno
%type <byteval> ixbpool
%type <byteval> ixbpoolspec
%type <byteval> bpoolparm
%type <byteval> bufferpoolparm
%type <byteval> beforeafter
%type <byteval> locksize
%type <byteval> lockparm
%type <byteval> closeparm
%type <tbnam> tb2name
%type <tbnam> tb3name
%type <tbnam> table3name
```

```
%token ADD AFTER ALIAS ALL ANY ALTER AS AUDIT AUDITSPEC BEFORE BIT
%token BP0 BP1 BP2 BP32K
%token BUFFERPOOL CASCADE CATALOG CDL PCHAR CHANGES CHECK CLOSE
CLUSTER
%token COLLATERULE PCOMMENT COLUMN COLUMNS
%token COMMA CONSTANT CREATE CYL DATA DATABASE DATATYPE DATE DB2
%token DECIMAL DEFAULT DELETE
%token DEST DROP
%token DSETPASS EDITPROC
%token ERROR ERASE FIELDPROC FOR FOREIGN FLOAT
%token FREEPAGE GRAPHIC INTEGER INDEX IS K KEY LABEL
%token <strval> LGDELID
%token LOCATION
%token <strval>LOCID
%token LPAREN LOCKSIZE
%token <strval> LONGID
%token LONGVARCHAR LONGVARGRAPHIC MGPR MOVE
%token NAME NO NODSETPASS NOVCATPASS NONE NOT NULLB
%token <numval> NUMBER
%token NUMPARTS OF OFF ON
%token ORDERED ORGN OPTION
%token OWNER PAGE
%token PART
%token PCTFREE PERIOD PRIMARY PRIMARYTB PRIMARYVOL PRIQTY PROG
%token <strval> QUOTEDSTR
%token REFERENCETB RELEASE REMOVE REPLACE RESTRICT RPAREN SECQTY
SEGSIZE
%token SELECT SEMIC SEQUENCE SET SCPR
%token <strval> SHDELID
%token <strval> SHORTID
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
%token SMALLINT SRNM SRTY SSID STOGROUP SYNONYM SUBPAGES
%token TABLE TABLESPACE TIME TIMESTAMP
%token TRNM TRK TRTY TYPE UNIQUE UNITS WAS WITH VALIDPROC
%token VALUES VARCHAR VARGRAPHIC
%token VCAT VCATP VCATPASS VIEW
%token <strval> VOLSER
%token VOLUMES YES

%start program

%%

program
   : statement
   | program SEMIC statement
   | error
           {
            /* CLEAN UP PARSER DATA STRUCTURES FOR NEXT STMT */
            acmpclea ( &acmpdata );
           }
   | SEMIC
   ;

statement
   : CDL stnum stmttype
   | commandstmt
   ;

stnum
   : NUMBER
           {
            acmplineno = $1;
           }
   ;

stmttype
   : ALTER alterstmt
   | CREATE createstmt
   | DROP dropstmt
   | commentstmt
   | labelstmt
   ;

commandstmt
   : PROG stnum QUOTEDSTR
           {
            memset ( acmpdata.ptree->header.progname, ' ',
                   sizeof(CHAR56) - 1 );
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
          if ( $3.llen > sizeof(CHAR56) )
          {
           $3.llen = sizeof(CHAR56);
          }
          memcpy ( acmpdata.ptree->header.progname, $3.str,
                $3.llen );
          acmpdata.ptree->header.progname[sizeof(CHAR56) - 1] = '\0';
         }
 | ORGN stnum SSID shortid LOCATION shortid RELEASE QUOTEDSTR
         {
          acmpvtn ( acmpdata.ptree->header.origssid, $4 );
          acmpvtn ( acmpdata.ptree->header.origloc, $6 );
          memset ( acmpdata.ptree->header.dbrelease, ' ',
                sizeof( CHAR10 ) - 1 );
          if ( $8.llen > sizeof(CHAR10) )
          {
           $8.llen = sizeof(CHAR10);
          }
          memcpy ( acmpdata.ptree->header.dbrelease, $8.str,
                $8.llen );
          acmpdata.ptree->header.dbrelease[sizeof(CHAR10) - 1] =
                '\0';
         }

 | DEST stnum SSID shortid LOCATION shortid
         {
          acmpvtn ( acmpdata.ptree->header.destssid, $4 );
          acmpcid ( acmpdata.ptree->header.destloc, $6 );
         }

 | SCPR stnum shortid PERIOD longid TYPE longid
         {
          acmpvtn ( acmpdata.ptree->header.scopequal, $3 );
          acmpcid ( acmpdata.ptree->header.scopename, $5 );
          acmpvtn ( acmpdata.ptree->header.scopetype, $7 );
         }

 | MGPR stnum shortid PERIOD longid
         {
          acmpvtn ( acmpdata.ptree->header.migprofqual, $3 );
          acmpcid ( acmpdata.ptree->header.migprofname, $5 );
         }
 | SRNM stnum QUOTEDSTR
         {
          memset ( acmpdata.ptree->header.sourcename, ' ',
                sizeof ( CHAR56 ) - 1 );
          if ( $3.llen > sizeof(CHAR56) )
          {
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
                   $3.llen = sizeof(CHAR56);
                  }
                  memcpy ( acmpdata.ptree->header.sourcename,
                        $3.str, $3.llen );
                  acmpdata.ptree->header.sourcename[sizeof(CHAR56) - 1] =
                        '\0';
                }

         | TRNM stnum QUOTEDSTR
                {
                  memset ( acmpdata.ptree->header.trgtname, ' ',
                        sizeof(CHAR56) - 1 );
                  if ( $3.llen > sizeof(CHAR56) )
                  {
                   $3.llen = sizeof(CHAR56);
                  }
                  memcpy ( acmpdata.ptree->header.trgtname,
                        $3.str, $3.llen );
                  acmpdata.ptree->header.trgtname[sizeof(CHAR56) - 1] =
                        '\0';
                }
         | SRTY stnum tgttype
                {
                  acmpvtn( acmpdata.ptree->header.sourcetype, $3 );
                }
         | TRTY stnum tgttype
                {
                  acmpvtn ( acmpdata.ptree->header.trgttype, $3 );
                }
         | TIME stnum QUOTEDSTR
                {
                  memset ( acmpdata.ptree->header.timestamp, ' ',
                        sizeof ( CHAR28 ) - 1 );
                  if ( $3.llen > sizeof(CHAR28) )
                  {
                   $3.llen = sizeof(CHAR28);
                  }
                  memcpy ( acmpdata.ptree->header.timestamp, $3.str,
                        $3.llen );
                  acmpdata.ptree->header.timestamp[sizeof(CHAR28) - 1] =
                        '\0';
                }
         ;

       tgttype
         : shortid
                { $$ = $1; }
         | shortid DB2 CATALOG
```

```
            { $$ = $1; }
    ;
alterstmt
    : altdatabase
    | alttablespace
    | alttable
    | altforeignkey
    | altindex
    | altview
    | altsynonym
    | altalias
    | altstogroup
    ;

createstmt
    : crdatabase
    | crtablespace
    | crtable
    | crforeignkey
    | crindex
    | crview
    | crsynonym
    | cralias
    | crstogroup
    ;

dropstmt
    : dpdatabase
    | dptablespace
    | dptable
    | dpforeignkey
    | dpindex
    | dpview
    | dpsynonym
    | dpalias
    | dpstogroup
    | dpcomment
    | dplabel
    ;


/*
**          ALTER DATABASE
*/

altdatabase
    : DATABASE shortid alterdblist
            {
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
                    acmp_reason = acmpdatb ( ACMG_ALTER, $2, &acmpdata );
                    if ( acmp_reason != ACMG_RC_OK )
                    {
                     ACMPCERROR;
                    }

                   }
    ;

alterdblist
    : alterdbparm
    | alterdblist alterdbparm
    ;

alterdbparm
    : newsname
                   {
                    acmp_reason = acmpsnam( ACMG_DBD, $1, &acmpdata );
                    if ( acmp_reason != ACMG_RC_OK )
                    {
                     ACMPCERROR;
                    }
                   }
    | ownerparm
              {
               acmp_reason = acmponam ( ACMG_DBD, $1, &acmpdata );
               if ( acmp_reason != ACMG_RC_OK )
               {
                ACMPCERROR;
               }
              }

    | createparm
    ;


/*
**          ALTER TABLESPACE
*/

alttablespace
    : TABLESPACE tsname tsparmlist
                   {
                    acmp_reason = acmptabs( ACMG_ALTER, &acmpdata );
                    if ( acmp_reason != ACMG_RC_OK )
                    {
                     ACMPCERROR;
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
                }
              }
     ;

tsparmlist
     : altertsparm
     | tsparmlist altertsparm
     ;

altertsparm
     : newsname
               {
                acmp_reason = acmpsnam( ACMG_TSD, $1, &acmpdata );
                if ( acmp_reason != ACMG_RC_OK )
                {
                 ACMPCERROR;
                }
               }

     | dbasename
               {
               /* ALLOCATE THE TABLESPACE OBJECT, IF NECESSARY */
               acmp_retcode = acmpaloc ( ACMG_TSD, &acmpdata );
               if ( acmp_retcode EQ ACMG_RC_OK )
               {
                /* SEE IF NEW DATABASE ALREADY SPEC'ED */
          if ( acmpdata.tabsp->ndbn_pv->data[0] EQ '' )
                {
           /* STORE THE NEW NAME PARAMETER */
           acmpcid ( acmpdata.tabsp->ndbn_pv, $1 );
                }
                else
                {
                     acmp_reason = ACMG_SPECERR;
                     ACMPCERROR;
                }
               }
               else
               {
                acmp_reason = ACMG_SPACE;
                ACMPCERROR;
               }
          }

    | partspec
    | tsparms
    ;
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
tsparms
   : bufferpoolparm
                   {

                    acmp_reason = acmpbfpl ( ACMG_TSD, $1, &acmpdata );
                    if ( acmp_reason != ACMG_RC_OK )
                    {
                     ACMPCERROR;
                    }
                   }
   | lockparm
                   {
                    acmp_reason = acmplksz( $1, &acmpdata );
                    if ( acmp_reason != ACMG_RC_OK )
                    {
                     ACMPCERROR;
                    }
                   }

   | closeparm
                   {
                    acmp_reason = acmpclr( $1, &acmpdata );
                    if ( acmp_reason != ACMG_RC_OK )
                    {
                     ACMPCERROR;
                    }
                   }

   | passwparm
                   {
                    acmp_reason = acmppas( $1, &acmpdata );
                    if ( acmp_reason != ACMG_RC_OK )
                    {
                     ACMPCERROR;
                    }
                   }

   | segparm
                   {

                    acmp_reason = acmpseg( $1, &acmpdata );
                    if ( acmp_reason != ACMG_RC_OK )
                    {
                     ACMPCERROR;
                    }
                   }

   | eraserule
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
                {

                 acmp_reason = acmperas( $1, &acmpdata );
                 if ( acmp_reason != ACMG_RC_OK )
                 {
                  ACMPCERROR;
                 }
                }
          | ownerparm
                {
                 acmp_reason = acmponam ( ACMG_TSD, $1, &acmpdata );
                 if ( acmp_reason != ACMG_RC_OK )
                 {
                  ACMPCERROR;
                 }
                }
          | partsparm
                {
                 acmp_reason = acmppts( $1, &acmpdata );
                 if ( acmp_reason != ACMG_RC_OK )
                 {
                  ACMPCERROR;
                 }
                }


          ;


/*
**        TABLESPACE PARTS SPECIFICATION
*/

partspec
    : alterpart
    | addpart
    | droppart
    ;

alterpart
    : ALTER PART NUMBER LPAREN partparmlist RPAREN
                    {
                acmp_reason = acmpcpt( ACMG_ALTER, ACMG_ALTER,
                                $3, &acmpdata );
                    if ( acmp_reason != ACMG_RC_OK )
                    {
                     ACMPCERROR;
                    }
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
                  }
    ;

addpart
    : ADD PART beforeafter NUMBER LPAREN partparmlist RPAREN
                  {
                   /* ADD THE PARAMETERS TO THE PARTS STRUCTURE */
            acmpdata.parts->drct = $3;
            acmpdata.parts->trgt = $4;

            acmp_reason = acmpcpt( ACMG_CREATE, ACMG_ALTER,
                          $4, &acmpdata );
                   if ( acmp_reason != ACMG_RC_OK )
                   {
                    ACMPCERROR;
                   }

                   /* RESET PART NUMBER TO ZERO */
                   acmpdata.tabsp->part_p->nmbr = 0;
                  }

    ;

droppart
    : DROP PART NUMBER
                 {
                  /* ALLOCATE A PARTS OBJECT */
                  acmp_retcode = acmpaloc ( ACMG_PTD, &acmpdata );
                  if ( acmp_retcode EQ ACMG_RC_OK )
                  {
            acmp_reason = acmpcpt( ACMG_DROP, ACMG_ALTER,
                          $3, &acmpdata );
                   if ( acmp_reason != ACMG_RC_OK )
                   {
                    ACMPCERROR;
                   }
                  }
                 }
    ;

partparmlist
    : partparm
    | partparmlist partparm
    ;

partparm
    : vcatparms
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
| stogroupparm
            {
             acmp_reason = acmpstgp ( ACMG_PTD, $1, &acmpdata );
             if ( acmp_reason  != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }


| priqtyparm
            {
             acmp_reason = acmpprq( $1, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }

| secqtyparm
            {
             acmp_reason = acmpsec( $1, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }

| pctfreeparm
            {
             acmp_reason = acmppcfr( $1, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }

| freepageparm
            {
             acmp_reason = acmpfrpg( $1, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
    ;

/*
**              ALTER TABLE
*/

alttable
    : TABLE tb2name altertblist
    ;

altertblist
    : altertbparm
    | altertblist altertbparm
    ;

altertbparm
    : tbname2parm
    | ownerparm
    | dbasename
    | tsparm
    | editparm
    | validparm
    | auditparm
    | altercolumn
    | prikeys
    ;

/*
**          ALTER TABLE COLUMNS
*/

altercolumn
    : ALTER COLUMN longid LPAREN altcolspec RPAREN
    | DROP COLUMN longid
    | ADD COLUMN longid beforeafter longid LPAREN colspec RPAREN
    ;

altcolspec
    : altcolspecparm
    | altcolspec altcolspecparm
    ;

altcolspecparm
    : seqparm
    | typeparm
    | forbitspec
    | fieldprocspec
    | nullrule
    | newlname
    | moveparm
    ;

colspec
    : colspecparm
    | colspec colspecparm
    ;

colspecparm
    : seqparm
    | typeparm
    | forbitspec
    | fieldprocspec
    | nullrule
    ;

moveparm
    : MOVE beforeafter longid
    ;
/*
**          ALTER TABLE PRIMARY KEYS
*/

prikeys
    : dropkey
    | addkey
    | alterkey
    ;

dropkey
    : DROP PRIMARY KEY
    ;

addkey :
    ADD PRIMARY KEY LPAREN keycolumnlist RPAREN
    ;

keycolumnlist
    : longid
    | keycolumnlist COMMA longid
    ;
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
alterkey
    : ALTER PRIMARY KEY LPAREN keycolumnlist RPAREN
    ;


/*
**          ALTER FOREIGN KEY
*/


altforeignkey
    : FOREIGN KEY fknamespec FOR tb2name alterclauses
    ;


alterclauses
    : alterfkparm
    | alterclauses alterfkparm
    ;

alterfkparm
    : newfkname
    | referencetb
    | fkcollist
    | colcollist
    | deletespec
    | PRIMARYTB tb2name
    ;

newfkname
    : NAME NONE
    | NAME shortid
    ;

colnamelist
    : longid
    | colnamelist COMMA longid
    ;

colcollist
    : WAS COLUMNS LPAREN colnamelist RPAREN
    ;

fkcollist
    : COLUMNS LPAREN colnamelist RPAREN
    :
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
/*
**          ALTER INDEX
*/

altindex
    : INDEX ixname alterixparms
    ;

alterixparms
    : alterixparm
    | alterixparms alterixparm
    ;

alterixparm
    : table2name
    | newlname
    | ownerparm
    | uniquerule
    | clusterspec
    | subpages
    | ixbpoolspec
    | ixpartspec
    | alterixcol
    | closeparm
    | passwparm
    ;

/*
**          INDEX COLUMN SPECS
*/

alterixcol
    : ADD COLUMN longid beforeafter longid addixcolparms
    | DROP COLUMN longid
    | ALTER COLUMN longid LPAREN altixcolparms RPAREN
    ;

altixcolparms
    : altixcolparm
    | altixcolparms altixcolparm
    ;

altixcolparm
    : commonixcolparm
    | newlname
    ;
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
addixcolparms
   : commonixcolparm
   | addixcolparms commonixcolparm
   ;

commonixcolparm
   : COLLATERULE
   | eraserule
   ;

ixpartspec
   : ADD PART NUMBER ixpartparms
   | ALTER PART NUMBER ixpartparms
   | DROP PART NUMBER
   ;

ixpartparms
   : ixpartparm
   | ixpartparms ixpartparm
   ;

ixpartparm
   : VALUES LPAREN constlist RPAREN
   | vcatparms
   | stogroupparm
   | freepageparm
   | pctfreeparm
   | priqtyparm
   | secqtyparm
   | eraserule
   ;


/*
**          ALTER VIEW
*/

altview
   : VIEW tb2name altvwparms
   ;

altvwparms
   : altvwparm
   | altvwparms altvwparm
   ;
```

```
                                   Change Definition Language
                                     BNF Grammer Definition
                                      With C Action Routines
/* this part needs work by JEO, what parameters should be provided for
   constructing view text ?? */

altvwparm
    : addvwcol
    | delvwcol
    | altervwcol
    | checkopt
    | viewtext
    | newlname
    | ownerparm
/*  | NO OWNER                      ????? */
    ;

addvwcol
    : ADD COLUMN longid beforeafter longid
    ;

altervwcol
    : ALTER COLUMN longid newlname
    ;

delvwcol
    : DROP COLUMN longid
    ;

/*
**      ALTER SYNONYM
*/


altsynonym
    : SYNONYM tb2name altersynparms
            {
             /* STORE THE SYNONYM NAME */
             acmpcid ( acmpdata.syns->name_pv, $2->tbna );
             acmpcid ( acmpdata.syns->ownr_pv, $2->tbow );
             acmpdata.syns->clvl = ACMG_ALTER;
             acmp_reason = acmpsyns ( acmpdata.ptree, acmpdata.syns );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }
    ;

altersynparms
    : altersynparm
    | altersynparms altersynparm
    ;

altersynparm
    : TABLE tb2name
            {
             acmp_reason = acmpaloc ( ACMG_SYD, &acmpdata );
             if ( acmp_reason EQ ACMG_RC_OK )
             {
              acmpcid ( acmpdata.syns->tbow_pv, $2->tbow );
              acmpcid ( acmpdata.syns->tbna_pv, $2->tbna );
             }
             else
             {
              ACMPCERROR;
             }
            }
    | newlname
            {
             acmp_reason = acmpsnam ( ACMG_SYD, $1, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }
    | ownerparm
            {
             acmp_reason = acmponam ( ACMG_SYD, $1, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }
    ;


/*
**      ALTER ALIAS
*/

altalias
    : ALIAS tb2name altalparms
            {
             /* COPY THE ALIAS NAME */
             acmpcid ( acmpdata.alias->ownr_pv, $2->tbow );
             acmpcid ( acmpdata.alias->name_pv, $2->tbna );
             acmpdata.alias->clvl = ACMG_ALTER;
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
              /* STORE THE ALIAS NODE IN THE PARSE TREE */
              acmp_reason = acmpalia( acmpdata.ptree, acmpdata.alias );
              if ( acmp_reason != ACMG_RC_OK )
              {
               ACMPCERROR;
              }
             }
     ;

altalparms
    : altalparm
    | altalparms altalparm
    ;

altalparm
    : table3name
          {
           acmp_reason = acmpaloc ( ACMG_ALD, &acmpdata );
           if ( acmp_reason EQ ACMG_RC_OK )
           {
            if ( $1->tbloc != NULL )
            {
             acmpcid ( acmpdata.alias->tblc_pv, $1->tbloc );
            }
            acmpcid ( acmpdata.alias->tbow_pv, $1->tbow );
            acmpcid ( acmpdata.alias->tbna_pv, $1->tbna );
           }
           else
           {
            acmp_reason = ACMG_SPACE;
            ACMPCERROR;
           }
           acmgfree ( $1 );
          }
    | ownerparm
          {
           acmp_reason = acmponam ( ACMG_ALD, $1, &acmpdata );
           if ( acmp_reason != ACMG_RC_OK )
           {
            ACMPCERROR;
           }
          }
    | newlname
          {
           acmp_reason = acmpsnam ( ACMG_ALD, $1, &acmpdata );
           if ( acmp_reason != ACMG_RC_OK )
           {
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
                ACMPCERROR;
               }
              }
    ;


/*
**      ALTER STOGROUP
*/

altstogroup
    : STOGROUP stogroupname altstogroupparms
              {
               acmp_reason = acmpstog ( ACMG_ALTER, &acmpdata );
               if ( acmp_reason != ACMG_RC_OK )
               {
                ACMPCERROR;
               }

              }

    ;

altstogroupparms
    : altstogroupparm
    | altstogroupparms altstogroupparm
    ;

altstogroupparm
    : newsname
               {
                acmp_reason = acmpsnam( ACMG_SGD, $1, &acmpdata );
                if ( acmp_reason != ACMG_RC_OK )
                {
                 ACMPCERROR;
                }
               }

    | ownerparm
               {

                acmp_reason = acmponam( ACMG_SGD, $1, &acmpdata );
                if ( acmp_reason != ACMG_RC_OK )
                {
                 ACMPCERROR;
                }
               }
    | VOLUMES LPAREN vollist RPAREN
            {
             acmp_reason = acmpvlst( ACMG_SGD, ACMG_REPLACE, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }

    | stogvcat
    ;

stogvcat
    : VCAT shortid
            {
             acmp_reason = acmpvcat( ACMG_SGD, $2, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }

    | vcatpass
            {

             acmp_reason = acmpvps( ACMG_SGD, $1, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }
    ;


/*
**      CREATE DATABASE
*/

crdatabase
    : DATABASE shortid createlist
            {
             acmp_retcode = acmpdatb ( ACMG_CREATE, $2, &acmpdata );
             if ( acmp_retcode != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
            }
      | DATABASE shortid
            {
             acmp_retcode = acmpaloc ( ACMG_DBD, &acmpdata );
             if ( acmp_retcode EQ ACMG_RC_OK )
             {

              acmp_retcode = acmpdatb ( ACMG_CREATE, $2, &acmpdata );
              if ( acmp_retcode != ACMG_RC_OK )
              {
               ACMPCERROR;
              }
             }
             else
             {
              acmp_retcode = ACMG_SPACE;
              ACMPCERROR;
             }
                 }

      ;

createlist
      : createparms
      | createlist createparms
      ;

createparms
      : ownerparm
            {
             acmp_reason = acmponam ( ACMG_DBD, $1, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }
      | createparm
      ;


createparm
      : stogroupparm
                  {
                   acmp_reason = acmpstgp ( ACMG_DBD, $1, &acmpdata );
                   if ( acmp_reason != ACMG_RC_OK )
                   {
                    ACMPCERROR;
                   }
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
                }


  | bufferpoolparm
                {
                 acmp_reason = acmpbfpl ( ACMG_DBD, $1, &acmpdata );
                 if ( acmp_reason != ACMG_RC_OK )
                 {
                  ACMPCERROR;
                 }
                }


  ;


/*
**         CREATE TABLESPACE
*/

crtablespace
  : TABLESPACE tsname createtslist
          {
           acmp_reason = acmgdfp( acmpdata.tabsp );
           if ( acmp_reason != ACMG_RC_OK )
           {
            ACMPCERROR;
           }

                acmp_reason = acmptabs( ACMG_CREATE, &acmpdata );
                if ( acmp_reason != ACMG_RC_OK )
                {
                 ACMPCERROR;
           }
               }

  | TABLESPACE tsname
               {
                acmp_reason = acmptabs( ACMG_CREATE, &acmpdata );
                if ( acmp_reason != ACMG_RC_OK )
                {
                 ACMPCERROR;
                }
               }
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
createtslist
    : createtsparm
    | createtslist createtsparm
    ;

createtsparm
    : tsparms
    | crpartspec
    ;


crpartspec
    : PART NUMBER LPAREN partparmlist RPAREN
            {
             acmp_reason = acmpcpt( ACMG_CREATE, ACMG_CREATE,
                         $2, &acmpdata );
                  if ( acmp_reason != ACMG_RC_OK )
                  {
                   ACMPCERROR;
                  }
                 }


    ;

/*
**          CREATE TABLE
*/

crtable
    : TABLE tb2name createtblist
    ;

createtblist
    : createtbparm
    | createtblist createtbparm
    ;

createtbparm
    : dbasename
    | tsparm
    | editparm
    | validparm
    | auditparm
    | crprikey
    | createcolspec
    ;
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
/*
**          CREATE TABLE COLUMNS SPEC
*/

createcolspec
    : COLUMNS LPAREN collist RPAREN
    ;

collist
    : coldef
    | collist COMMA coldef
    ;

coldef
    : longid colparmlist
    ;

colparmlist
    : colparm
    | colparmlist colparm
    ;

colparm
    : typeparm
    | FOR BIT DATA
    | fieldprocspec
    | nullrule
    ;

/*
**          CREATE TABLE PRIMARY KEY
*/

crprikey
    : PRIMARY KEY LPAREN keycolumnlist RPAREN
    ;


/*
**          CREATE FOREIGN KEY
*/

crforeignkey
    : FOREIGN KEY fknamespec FOR tb2name addclauses
    ;

addclauses
    : addfkparm
    | addclauses addfkparm
    ;

addfkparm
    : referencetb
    | fkcollist
    | deletespec
    ;

/*
**          CREATE INDEX
*/

crindex
   : CREATE INDEX ixname createixparms
   ;

createixparms
   : createixparm
   | createixparms createixparm
   ;

createixparm
   : table2name
   | uniquerule
   | clusterspec
   | subpages
   | ixbpoolspec
   | crixpart
   | createixcol
   | closeparm
   | passwparm
   ;


/*
**          CREATE INDEX COLUMNS
*/

createixcol
   : COLUMNS LPAREN ixcollist RPAREN
   ;

ixcollist
   : longid
   | addixcolparms COMMA longid
   ;
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
/*
**        CREATE INDEX PARTS
*/

crixpart
   : PART NUMBER ixpartparms
   ;



/*
**        CREATE VIEW
*/

crview
   : VIEW tb2name crvwparms
   ;

crvwparms
   : crvwparm
   | crvwparms crvwparm
   ;

crvwparm
   : coldesc
   | checkopt
   | viewtext
   ;


/*
**        CREATE SYNONYM
*/

crsynonym
   : CREATE SYNONYM tb2name TABLE tb2name
        {
        /* ALLOCATE THE SYNONYM TEMPORARY OBJECT */
        acmp_reason = acmpaloc ( ACMG_SYD, &acmpdata );
        if ( acmp_reason EQ ACMG_RC_OK )
        {
         /* STORE SYNONYM PARAMETERS */
         acmpcid ( acmpdata.syns->name_pv, $3->tbna );
         acmpcid ( acmpdata.syns->ownr_pv, $3->tbow );
         acmpcid ( acmpdata.syns->tbna_pv, $5->tbna );
         acmpcid ( acmpdata.syns->tbow_pv, $5->tbow );
         acmpdata.syns->clvl = ACMG_CREATE;
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
                /* STORE NODE IN PARSE TREE */
                acmp_reason = acmpsyns ( acmpdata.ptree, acmpdata.syns );
                if ( acmp_reason != ACMG_RC_OK )
                {
                 ACMPCERROR;
                }
               }
               else
               {
                acmp_reason = ACMG_SPACE;
               }
              }
     ;


/*
**         CREATE ALIAS
*/

cralias
   : ALIAS tb2name table3name
          {
           /* ALLOCATE THE ALIAS TEMPORARY OBJECT */
           acmp_reason = acmpaloc ( ACMG_ALD, &acmpdata );
           if ( acmp_reason EQ ACMG_RC_OK )
           {
            /* STORE ALIAS PARAMETERS */
            acmpcid ( acmpdata.alias->name_pv, $2->tbna );
            acmpcid ( acmpdata.alias->ownr_pv, $2->tbow );
            acmpcid ( acmpdata.alias->tbna_pv, $3->tbna );
            acmpcid ( acmpdata.alias->tbow_pv, $3->tbow );
            acmpcid ( acmpdata.alias->tblc_pv, $3->tbloc );
            acmpdata.alias->clvl = ACMG_CREATE;

            /* STORE THE ALIAS OBJECT IN THE PARSE TREE */
            acmp_reason = acmpalia ( acmpdata.ptree,
                          acmpdata.alias );
            if ( acmp_reason != ACMG_RC_OK )
            {
             ACMPCERROR;
            }
           }
           else
           {
            acmp_reason = ACMG_SPACE;
            ACMPCERROR;
           }
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
        }
  ;

/*
**       CREATE STOGROUP
*/

crstogroup
   : STOGROUP stogroupname stogroupparms
          {
           acmp_reason = acmpstog ( ACMG_CREATE, &acmpdata );
           if ( acmp_reason != ACMG_RC_OK )
           {
            ACMPCERROR;
           }

          }
  ;

stogroupparms
   : stogroup
   | stogroupparms stogroup
   ;

stogroup
   : stogvcat
   | VOLUMES LPAREN vollist RPAREN
                {
          acmp_reason = acmpvlst( ACMG_SGD, ACMG_REPLACE, &acmpdata );
                 if ( acmp_reason != ACMG_RC_OK )
                 {
                  ACMPCERROR;
                 }
          }
   | ownerparm
          {
           acmp_reason = acmponam( ACMG_SGD, $1, &acmpdata );
           if ( acmp_reason != ACMG_RC_OK )
           {
            ACMPCERROR;
           }
          }


  ;

/*
**         COMMENT ON
```

```
                    Change Definition Language
                    BNF Grammer Definition
                    With C Action Routines
*/

commentstmt
   : PCOMMENT ON objectcomm
   ;

objectcomm
   : TABLE tb2name IS QUOTEDSTR
   | ALIAS tb2name IS QUOTEDSTR
   | COLUMN OF tb2name LPAREN colcomlist RPAREN
   ;

colcomlist
   : columncom
   | colcomlist COMMA columncom
   ;

columncom
   : longid IS QUOTEDSTR
   ;


/*
**              LABEL ON
*/

labelstmt
   : LABEL ON objectcomm
   ;


/*
**          DROP OBJECTS
*/

dpdatabase
    : DATABASE shortid
                {
                 acmp_retcode = acmpaloc( ACMG_DBD, &acmpdata );
                 if ( acmp_retcode EQ ACMG_RC_OK )
                 {
                  acmp_reason = acmpdatb ( ACMG_DROP, $2, &acmpdata );
                  if ( acmp_reason != ACMG_RC_OK )
                  {
                       ACMPCERROR;
                  }
                 }
                 else
```

```
                      {
                       acmp_reason = ACMG_SPACE;
                       ACMPCERROR;

                      }

                    }
    ;

dptablespace
    : TABLESPACE tsname
                    {

                     acmp_reason = acmptabs( ACMG_DROP, &acmpdata );
                     if ( acmp_reason != ACMG_RC_OK )
                     {
                      ACMPCERROR;
                     }
                    }
    ;

dptable
    : TABLE tb2name
    ;

dpforeignkey
    : FOREIGN KEY fknamespec FOR dpfkparms
    ;

dpfkparms
    : referencetb
    | colcollist
    ;

dpview
    : VIEW tb2name
    ;

dpindex
    : INDEX ixname table2name
    ;

dpsynonym
    : SYNONYM longid
              {
               acmp_retcode = acmpaloc ( ACMG_SYD, &acmpdata );
               if ( acmp_retcode EQ ACMG_RC_OK )
               {
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
                /* STORE GLOBAL AUTHID AS OWNER */
                acmpownr ( acmpdata.syns->ownr_pv, &acmpdata );

                /* STORE THE NAME AND ADD TO PARSE TREE */
                acmpdata.syns->clvl = ACMG_DROP;
                acmpcid ( acmpdata.syns->name_pv, $2 );
                acmp_reason = acmpsyns ( acmpdata.ptree,
                              acmpdata.syns );
                if ( acmp_reason != ACMG_RC_OK )
                {
                 ACMPCERROR;
                }
               }
               else
               {
                acmp_reason = ACMG_SPACE;
                ACMPCERROR;
               }
              }

       ;

dpalias
   : ALIAS tb2name
              {
               acmp_retcode = acmpaloc ( ACMG_ALD, &acmpdata );
               if ( acmp_retcode EQ ACMG_RC_OK )
               {
                /* STORE THE NAME */
                acmpcid ( acmpdata.alias->name_pv, $2->tbna );
                acmpcid ( acmpdata.alias->ownr_pv, $2->tbow );
                acmpdata.alias->clvl = ACMG_DROP;

                /* STORE THE OBJECT IN PARSE TREE */
                acmp_reason = acmpalia ( acmpdata.ptree,
                              acmpdata.alias );
                if ( acmp_reason != ACMG_RC_OK )
                {
                 ACMPCERROR;
                }
               }
               else
               {
                acmp_reason = ACMG_SPACE;
                ACMPCERROR;
               }
              }
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
  ;

dpstogroup
  : STOGROUP stogroupname
              {
               acmp_retcode = acmpaloc( ACMG_DBD, &acmpdata );
               if ( acmp_retcode EQ ACMG_RC_OK )
               {

                acmp_reason = acmpstog ( ACMG_DROP, &acmpdata );
                if ( acmp_reason != ACMG_RC_OK )
                {
                    ACMPCERROR;
                }

              }
              else
              {
               acmp_reason = ACMG_SPACE;
               ACMPCERROR;

              }

             }

  ;

dpcomment
   : PCOMMENT ON dpobject
   ;

dpobject
   : TABLE tb2name
   | ALIAS tb2name
   | COLUMN OF tb2name LPAREN colnamelist RPAREN
   ;

dplabel
   : LABEL ON dpobject
   ;


/*
**        MISCELLANEOUS COMMAND PARAMETERS
*/
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
auditparm
    : AUDIT AUDITSPEC
    ;

beforeafter
    : BEFORE                        { $$ = ACMG_BEFORE; }
    | AFTER                         { $$ = ACMG_AFTER; }




    ;

bufferpoolparm
    : BUFFERPOOL bpoolparm          { $$ = $2; }

    ;

bpoolparm
    : ixbpool                       { $$ = $1; }
    | BP32K                         { $$ = ACMG_BP32K; }

    ;

checkopt
   : CHECK OPTION yesorno

   ;

closeparm
    : CLOSE yesorno             { $$ = $2; }

    ;

clusterspec
   : CLUSTER yesorno

    ;

coldesc
   : COLUMN longid NUMBER
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
constlist
    : CONSTANT
    | constlist CONSTANT
    ;


datatype
    : PCHAR LPAREN NUMBER RPAREN
    | INTEGER
    | FLOAT LPAREN NUMBER RPAREN
    | LONGVARCHAR
    | VARCHAR LPAREN NUMBER RPAREN
    | SMALLINT
    | GRAPHIC LPAREN NUMBER RPAREN
    | VARGRAPHIC LPAREN NUMBER RPAREN
    | LONGVARGRAPHIC
    | DATE
    | TIME
    | TIMESTAMP
    | DECIMAL LPAREN NUMBER COMMA NUMBER RPAREN
    | DECIMAL LPAREN NUMBER RPAREN
    ;

dbasename
    : DATABASE shortid            { $$ = $2; }

    ;

deletespec
    : ON DELETE deleterule

    ;

deleterule
    : CASCADE
    | DELETE
    | SET NULLB
    ;


empty
    :
    ;

editparm
    : EDITPROC shortid
    | NO EDITPROC
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
    ;

eraserule
    : ERASE yesorno            { $$ = $2; }

    ;

fieldprocspec
    : FIELDPROC shortid
    | FIELDPROC shortid constlist
    ;

fknamespec
    : empty
    | shortid
    ;

forbit
    : FOR BIT DATA
    ;

forbitoff
    : forbit OFF
    ;

forbitspec
    : forbit
    | forbitoff
    ;

freepageparm
    : FREEPAGE NUMBER
            {
             $$ = $2;
             CHKRANGE( $$, 0, ACMG_MAXFRPG, ACMG_FREEPRANGE );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }

    ;

ixbpool
    : BP0              { $$ = ACMG_BP0; }
    | BP1              { $$ = ACMG_BP1; }
    | BP2              { $$ = ACMG_BP2; }
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
   ;

ixbpoolspec
   : BUFFERPOOL ixbpool          { $$ = $2; }

   ;

ixname
   : tb2name
   ;

lockparm
   : LOCKSIZE locksize           { $$ = $2; }

   ;

locksize
   : ANY               { $$ = ACMG_ANY; }
   | PAGE               { $$ = ACMG_PAGE; }
   | TABLE               { $$ = ACMG_TABLE; }
   | TABLESPACE               { $$ = ACMG_TABLESPACE; }

   ;

longid
  : LONGID
           {
            $$ = acmpalid ( &($1), 0, TRUE );
            if ( $$ EQ NULL )
            {
             acmp_reason = ACMG_SPACE;
             ACMPCERROR;
            }
           }
  | locationid
           { $$ = $1; }
  | shortid
           { $$ = $1; }

  | LGDELID
           {
            $$ = acmpalid ( &($1), 0, FALSE );
            if ( $$ EQ NULL )
            {
             acmp_reason = ACMG_SPACE;
             ACMPCERROR;
            }
            }

    ;

newlname
    : NAME longid
                            { $$ = $2; }
    ;

newsname
    : NAME shortid              { $$ = $2; }
    ;

nullrule
    : NOT NULLB
    | NOT NULLB WITH DEFAULT
    | NULLB
    ;

ownerparm
    : OWNER shortid             { $$ = $2; }
    ;

partsparm
    : NUMPARTS NUMBER
            {
             $$ = $2;
             CHKRANGE( $$, 0, ACMG_MAXPARTS, ACMG_PARTNORANGE );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }

    ;

passwparm
    : DSETPASS shortid          { $$ = $2; }
    | NODSETPASS                { $$ = NULL; }
    ;

pctfreeparm
    : PCTFREE NUMBER
            {
             $$ = $2;
             CHKRANGE( $$, 0, ACMG_MAXPCFR, ACMG_PCTFRANGE );
             if ( acmp_reason != ACMG_RC_OK )
             {
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
            ACMPCERROR;
           }
          }
      ;

   priqtyparm
      : PRIQTY NUMBER
           {
            $$ = $2;
            CHKRANGE( $$, ACMG_MINPRQT, ACMG_MAXPRQT, ACMG_PRQTRANGE );
            if ( acmp_reason != ACMG_RC_OK )
            {
             ACMPCERROR;
            }
           }
      ;

   referencetb
      : REFERENCETB tb2name
      ;

   segparm
      : SEGSIZE NUMBER
           {
            $$ = $2;
            if ( $$ != 0 )
            {
             if ( ($$ < ACMG_MINSEGSZ) ||
                 ($$ > ACMG_MAXSEGSZ) )
             {
              acmp_reason = ACMG_SEGSZERR;
              ACMPCERROR;
             }
             if ( ($$ % 4) != 0 )
             {
              acmp_reason = ACMG_SEGSZERR;
              ACMPCERROR;
             }
            }
           }
      ;

   seqparm
      : SEQUENCE NUMBER
      ;

   secqtyparm
```

Change Definition Language
BNF Grampner Definition
With C Action Routines

```
    : SECQTY NUMBER
            {
             $$ = $2;
             CHKRANGE( $$, ACMG_MINSCQT, ACMG_MAXSCQT, ACMG_SCQTRANGE
);
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }
    ;

locationid
    : LOCID
            {
             $$ = acmpalid ( &($1), ACMG_LCTNID, TRUE );
             if ( $$ EQ NULL )
             {
              acmp_reason = ACMG_SPACE;
              ACMPCERROR;
             }
            }
    ;

shortid
    : SHORTID
            {
             $$ = acmpalid ( &($1), ACMG_SHORTIDLEN, TRUE );
             if ( $$ EQ NULL )
             {
              acmp_reason = ACMG_SPACE;
              ACMPCERROR;
             }
            }

    | SHDELID
            {
             $$ = acmpalid ( &($1), ACMG_SHORTIDLEN, FALSE );
             if ( $$ EQ NULL )
             {
              acmp_reason = ACMG_SPACE;
              ACMPCERROR;
             }
            }

    | VOLSER
            {
             $$ = acmpalid ( &($1), ACMG_SHORTIDLEN, TRUE );
```

```
                    if ( $$ EQ NULL )
                    {
                     acmp_reason = ACMG_SPACE;
                     ACMPCERROR;
                    }
                   }

        ;

      stogroupparm
        : STOGROUP shortid              { $$ = $2; }
        ;

      stogroupname

        : shortid
              {

               acmp_reason = acmpsgnm( $1, NULL, &acmpdata );
               if ( acmp_reason != ACMG_RC_OK )
               {
                ACMPCERROR;
               }
              }

        ;

      subpages
        : SUBPAGES NUMBER
        ;

      table2name
        : TABLE tb2name
        ;

      table3name
        : TABLE tb3name
              { $$ = $2; }
        ;

      tbname2parm
        : NAME longid
        ;

      tb2name
        : shortid PERIOD longid
              {
               $$ = acmgaloc ( sizeof(struct ACMP_TN) );
```

```
             if ( $$ != NULL )
             {
              $$->tbow = $1;
              $$->tbna = $3;
             }
             else
             {
              acmp_reason = ACMG_SPACE;
              ACMPCERROR;
             }
            }
      | longid .
            {
             $$ = acmgaloc ( sizeof(struct ACMP_TN) );
             if ( $$ != NULL )
             {
              $$->tbna = $1;

              /* SET OWNER NAME TO GLOBAL AUTHID */
              $$->tbow = acmgaloc ( sizeof(VCHAR) + ACMG_SHORTIDLEN );
              if ( $$->tbow != NULL )
              {
               acmpownr ( $$->tbow, &acmpdata );
              }
              else
              {
               acmp_reason = ACMG_SPACE;
               ACMPCERROR;
              }

             }
             else
             {
              acmp_reason = ACMG_SPACE;
              ACMPCERROR;
             }
            }
      ;

    tb3name
      : locationid PERIOD shortid PERIOD longid
            {
             $$ = acmgaloc ( sizeof(struct ACMP_TN) );
             if ( $$ != NULL )
             {
              VCHAR *temp;

              /* REALLOCATE THE LOCATION PART OF NAME TO BE 16 BYTES*/
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
                temp = acmpalid ( (struct ACMP_ID *)$1, ACMG_LCTNID, FALSE );
                if ( temp != NULL )
                {
                 $$->tbloc = temp;
                 acmgfree ( $$->tbloc );
                }
                else
                {
                 acmp_reason = ACMG_SPACE;
                 ACMPCERROR;
                }
               $$->tbow = $3;
               $$->tbna = $5;
               if ( memcmp ( acmpdata.loc, $$->tbloc->data,
                       $$->tbloc->len ) EQ 0 )
               {
                /* SET LOCATION TO BLANKS IF IT IS LOCAL */
                memset ( $$->tbloc->data, ' ', $$->tbloc->len );
               }
              }
              else
              {
               acmp_reason = ACMG_SPACE;
               ACMPCERROR;
              }
             }
     | shortid PERIOD shortid PERIOD longid
             {
              $$ = acmgaloc ( sizeof(struct ACMP_TN) );
              if ( $$ != NULL )
              {

               VCHAR *temp;

               /* REALLOCATE THE LOCATION PART OF NAME TO BE 16 BYTES*/
               temp = acmpalid ( (struct ACMP_ID *)$1, ACMG_LCTNID, FALSE );
               if ( temp != NULL )
               {
                $$->tbloc = temp;
                acmgfree ( $$->tbloc );
               }
               else
               {
                acmp_reason = ACMG_SPACE;
                ACMPCERROR;
               }
               $$->tbow = $3;
               $$->tbna = $5;
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
          if ( memcmp ( acmpdata.loc, $$->tbloc->data,
                 $$->tbloc->len ) EQ 0 )
          {
           /* SET LOCATION TO BLANKS IF IT IS LOCAL */
           memset ( $$->tbloc->data, ' ', $$->tbloc->len );
          }
         }
         else
         {
          acmp_reason = ACMG_SPACE;
          ACMPCERROR;
         }
        }
  | tb2name
        {
         /* SET LOCATION ID TO BLANKS */
         $$->tbloc = acmgaloc ( sizeof(VCHAR) + ACMG_LCTNID );
         if ( $$->tbloc != NULL )
         {
          $$->tbloc->len = ACMG_LCTNID;
          memset ( $$->tbloc->data, ' ', ACMG_LCTNID );
         }
         else
         {
          acmp_reason = ACMG_SPACE;
          ACMPCERROR;
         }
        }
  ;

sname
  : shortid PERIOD shortid
        {

         acmp_reason = acmptnam( $3, $1, &acmpdata );
         if ( acmp_reason != ACMG_RC_OK )
         {
          ACMPCERROR;
         }
        }

  | shortid
        {
         VCHAR *temp;

         temp = acmgaloc ( sizeof(VCHAR) + ACMG_SHORTIDLEN );
         if ( temp EQ NULL )
         {
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
            acmp_reason = ACMG_SPACE;
            ACMPCERROR;
           }

           temp->len = ACMG_SHORTIDLEN - 1;
           memcpy ( &(temp->data), ACMG_DEFDBNAM, ACMG_SHORTIDLEN );
           acmp_reason = acmptnam( $1, temp, &acmpdata );
                  if ( acmp_reason != ACMG_RC_OK )
                  {
                   ACMPCERROR;
                  }
                 }
    ;

tsparm
    : TABLESPACE shortid
    ;

typeparm
    : datatype
    ;

uniquerule
    : UNIQUE yesorno
    ;

unit
    : K     { $$ = ACMG_K; }
    | CYL           { $$ = ACMG_CYL; }
    | TRK           { $$ = ACMG_TRK; }

    ;

validparm
    : NO VALIDPROC
    | VALIDPROC shortid
    ;

vcatparms
    : VCAT shortid LPAREN vcatparmlist RPAREN
            {

             /* ALLOCATE PARTS OBJECT */
             acmp_reason = acmpaloc ( ACMG_PTD, &acmpdata );
             if ( acmp_reason EQ ACMG_RC_OK )
             {
              acmp_reason = acmpvcat( ACMG_PVD, $2, &acmpdata );
              if ( acmp_reason != ACMG_RC_OK )
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
                {
                 ACMPCERROR;
                }
               }
               else
               {
                acmp_reason = ACMG_SPACE;
                ACMPCERROR;
               }
                    }


    ;

vcatparmlist
    : vcatparm
    | vcatparmlist vcatparm
    ;

vcatparm
    : VOLUMES LPAREN vollist RPAREN
            {

             acmp_reason = acmpvlst( ACMG_PVD, 0, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }

    | UNITS unit
            {
             acmp_reason = acmpunt( $2, &acmpdata );
             if ( acmp_reason != ACMG_RC_OK )
             {
              ACMPCERROR;
             }
            }

    | PRIMARYVOL volser
            {
             acmp_retcode = acmpaloc( ACMG_PVD, &acmpdata );
             if ( acmp_retcode EQ ACMG_RC_OK )
             {

              /* CHECK IF PRIMARY VOLUME WAS ALREADY SPEC'D */
              if ( acmpdata.vcats->prvl[0] == '' )
              {
               /* STORE PRIMARY VOLUME SPECIFICATION */
            acmpvtn ( acmpdata.vcats->prvl, $2 );

          }
          else
          {
            acmp_reason = ACMG_SPECERR;
            ACMPCERROR;
          }
        }
      }

  | ORDERED yesorno
        {
          acmp_retcode = acmpaloc( ACMG_PVD, &acmpdata );
          if ( acmp_retcode EQ ACMG_RC_OK )
          {

            /* CHECK IF ORDER WAS ALREADY SPEC'D */
            if ( acmpdata.vcats->ordr EQ ACMG_UNUSED )
            {
              /* STORE ORDER SPECIFICATION */
              acmpdata.vcats->ordr = $2;
            }
            else
            {
              acmp_reason = ACMG_SPECERR;
              ACMPCERROR;
            }
          }
        }

  | vcatpass
        {

          acmp_reason = acmpvps ( ACMG_PVD, $1, &acmpdata );

          if ( acmp_reason != ACMG_RC_OK )
          {
            ACMPCERROR;
          }
        }
  ;

vcatpass
    : VCATPASS shortid          { $$ = $2; }
    | NOVCATPASS                { $$ = NULL; }
    ;
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
viewtext
  : AS SELECT QUOTEDSTR
  ;

vollist
  : volser
          {

           acmp_reason = acmpvols( $1, &acmpdata );
           if ( acmp_reason != ACMG_RC_OK )
           {
            ACMPCERROR;
           }
          }

  | vollist COMMA volser
          {

           acmp_reason = acmpvols( $3, &acmpdata );
           if ( acmp_reason != ACMG_RC_OK )
           {
            ACMPCERROR;
           }
          }


  ;

volser
  : VOLSER
          {
           $$ = acmpalid ( &($1), ACMG_VOLSERLEN, TRUE );
           if ( $$ EQ NULL )
           {
            acmp_reason = ACMG_SPACE;
            ACMPCERROR;
           }
          }

  ;

yesorno
  : YES               { $$ = ACMG_YES; }
  | NO                { $$ = ACMG_NO; }

  ;
```

Change Definition Language
BNF Grammer Definition
With C Action Routines

```
%%

SHORT ACMG_DYN
acmpcerror ( SHORT lineno,
        SHORT reason,
        CHAR *s )

{
 return ( acmperx ( lineno, reason, s ) );

}
```

Claims

1. A method of altering an existing data structure in a database system at an operating node of said database system, comprising the steps of:
   a) receiving from another node a change indication specifying changes to be made in the existing data structure;
   b) checking said existing data structure for compatibility with said changes to be made;
   c) if said changes to be made are compatible with said existing data structure, then making said changes, or else signalling an error.

2. A method according to claim 1 wherein said change indication is in the format of a change definition language specified in accordance with Appendix A.

3. A method according to claim 1 wherein said data structure is defined by a catalog of tables, and said change definition language expresses changes to said data structure by changes to said catalog of tables.

4. A method according to claim 3 wherein said catalog of tables is the lowest "object" level of said database system.

5. A method according to claim 1 wherein said data structure includes tables, indexes and views, and said data structure is defined by a catalog including tables of tables, indexes and views.

6. A method according to claim 5 wherein said change indication is expressed in a change definition language for making changes in said catalog.

7. Apparatus for altering an existing data structure in a database system at an operating node of said database system, comprising:
   a) means for receiving from another node a change indication specifying changes to be made in the existing data structure;
   b) means for checking said existing data structure for compatibility with said changes to be made;
   c) means for making said changes if said changes to be made are compatible with said existing data structure, or, if not, signalling an error.

8. Apparatus according to claim 7 wherein said change indication is in the format of a change definition language specified in accordance with Appendix A.

9. Apparatus according to claim 7 wherein said data structure is defined by a catalog of tables, and said change definition language expresses changes to said data structure by changes to said catalog of tables.

10. Apparatus according to claim 9 wherein said catalog of tables is the lowest "object" level of said database system.

11. Apparatus according to claim 7 wherein said data structure includes tables, indexes and views, and said data structure is defined by a catalog including tables of tables, indexes and views.

12. Apparatus according to claim 11 wherein said change indication is expressed in a change definition language for making changes in said catalog.

13. Apparatus according to claim 12 wherein said node is one of a plurality of nodes of said system where changes to said database structure may be made.

14. Apparatus according to claim 13 wherein said change indication is a batch file of statements in said change definition language.

15. A method of altering an existing data structure in a database system at any one of a plurality of operating nodes of said database system, comprising the steps of:
a) receiving at one of said nodes from another of said nodes a change indication specifying changes to be made in the existing data structure at said one of said nodes;
b) checking said existing data structure at said one of said nodes for compatibility with said changes to be made;
c) if said changes to be made are compatible with said existing data structure, making said changes, or, if not, signalling an error.

16. A method according to claim 15 wherein said change indication is in the format of a change definition language specified in accordance with Appendix A.

17. A method according to claim 16 wherein said data structure is defined by a catalog of tables, and said change definition language expresses changes to said data structure by changes to said catalog of tables.

18. A method according to claim 17 wherein said catalog of tables is the lowest "object" level of said database system.

19. A method of operating a computer system, comprising the steps of:

generating and storing a data structure including a plurality of related tables of columns and rows of data values and indexes for said related tables to provide a relational database;

generating and storing a catalog of definitions of said data structure including definitions of said tables and indexes and views of said tables and indexes; said definitions including a set of tables of said tables, columns and indexes;

accessing said data structure and said catalog to provide a number of different views of data values in said data structure as defined by said catalog;

said steps of generating and storing said data structure and said catalog and said step of accessing being specified by a user by statements in a structured query language to define said tables, indexes, catalog, and views; and

altering said set of tables of said catalog to change at least some of said definition, in response to a set of statements of changes including statements for changing at least one of:

a data specification of a column,

the owner of a table, an index or a view,

the name of a column, an index or a view.

20. A method according to claim 19 wherein said set of statements is selected from the statements of Appendix A.

21. A method according to claim 19 wherein said structured query language is a special-purpose high-level programming language and said statements may be embedded in other different high-level programming languages and compiled to produce code for performing said steps of generating, storing and accessing.

22. Apparatus for operating a computer system, comprising:

means for generating and storing a data structure including a plurality of related tables of columns and rows of data values and indexes for said related tables to provide a relational database;

means for generating and storing a catalog of definitions of said data structure including tables defining said tables and indexes and views of said tables and indexes;

means for accessing said data structure and said catalog to provide a number of different views and reports of data values in said data structure as defined by said catalog;

said means for generating and storing said data structure and said catalog and said means for accessing being specified by a user by statements in a structured query language to define said tables, indexes, catalog, views and reports; and

means for altering said catalog to change at least some of said definition including a set of selected change statements for changing at least one of:

a data specification of a column,

the owner of a table, an index or a view,

the name of a column, an index or a view,

by changing said tables of said catalog.

23. Apparatus according to claim 22 wherein said change statements are selected from Appendix A.

24. Apparatus according to claim 22 wherein said structured query language is a special-purpose high-level programming language and said statements may be embedded in other different high-level programming languages and compiled to produce code for performing said steps of generating, storing and accessing.

25. A method of generating a set of change statements for changing the definition of the data structure of a database;

wherein said data structure includes a plurality of related tables of columns and rows and indexes for said related tables, and said definition is a catalog of said data structure including definitions of said tables and indexes and views of said tables and indexes, said catalog including a set of tables of said tables, columns and indexes; said catalog being specified by a user by statements in a data definition language to define said tables, indexes, and views;

comprising the steps of:

selecting from a library of statements compatible with said data definition language a plurality of statements to change said catalog to alter at least some of said definition including changing at least one of:

a data specification of a column,

the owner of a table, an index or a view,

the name of a column, an index or a view, and

comparing a representation of said plurality of statements to an existing instance of said catalog, and, if compatible with said existing instance, then generating said set of change statements.

26. A method according to claim 25 wherein said plurality of statements includes a statement for adding a column to a view.

27. A program storage device readable by a machine, tangibly embodying a program of instructions executable by a machine to perform the method of claim 25.

28. Apparatus for generating a set of change statements for changing the definition of the data structure of a database;

wherein said data structure includes a plurality of related tables of columns and rows and indexes for said related tables, and said definition is a catalog of said data structure including definitions of said tables and indexes and views of said tables and indexes, said catalog including a set of tables of said tables, columns and indexes; said catalog being specified by a user by statements in a structured query language to define said tables, indexes, and views;

comprising:

a set of statements selected from a library of statements compatible with said structured query language to change said set of tables of said catalog to alter at least some of said definition including changing at least one of:

a data specification of a column,

the owner of a table, an index or a view, or

the name of a column, an index or a view.

29. A method of describing a set of changes to an existing definition of a database, comprising the steps of:

saving a copy of said existing definition;

making a number of changes in said existing definition to produce a new definition of the database,

generating a set of change requests describing said number of changes in a change definition language, by selecting from a library of statements to alter at least some of said definition including changing at least one of:

a data specification of a column,

the owner of a table, an index or a view, or

the name of a column, an index or a view.

30. A method of describing a set of changes to an existing definition of a database, comprising the steps of:

saving a copy of said existing definition;

making a number of changes in said existing definition to produce a new definition of the database, generating a set of change requests describing said number of changes in a change definition language, by selecting from a library of statements as set forth in Appendix A.

31. A method of making changes in a data description catalog of a database, comprising the steps of:
a) receiving a list of change statements describing said changes, said change statements being selected from a library of change statements in the format of a change description language having a set of statements of changes including statements for changing:

a data specification of a column,

the owner of a table, an index or a view,

the name of a column, an index or a view;
b) generating from said list a set of changes to said data description, said set of changes being in a standard data description language for said database.

32. A method according to claim 31 wherein said database is DB2 and said data description language is SQL for DB2.

33. Apparatus for making changes in a data description catalog of a database, comprising:
a) means for receiving a list of change statements describing said changes, said change statements being selected from a library of change statements in the format of a change description language having a set of statements of changes including statements for changing:

a data specification of a column,

the owner of a table, an index or a view,

the name of a column, an index or a view;

b) means for generating from said list a set of changes to said data description, said set of changes being in a standard data description language for said database.

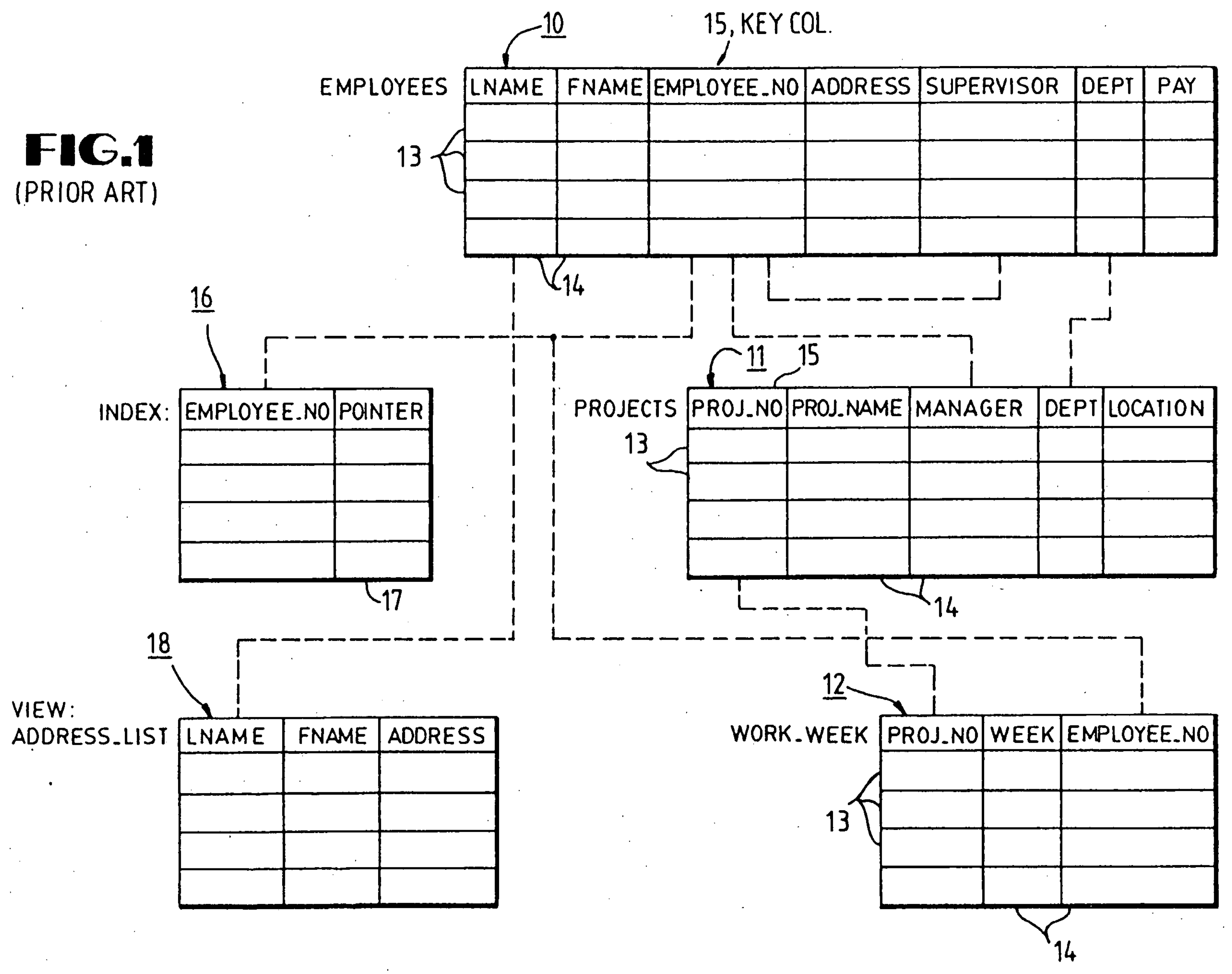


FIG.1
(PRIOR ART)

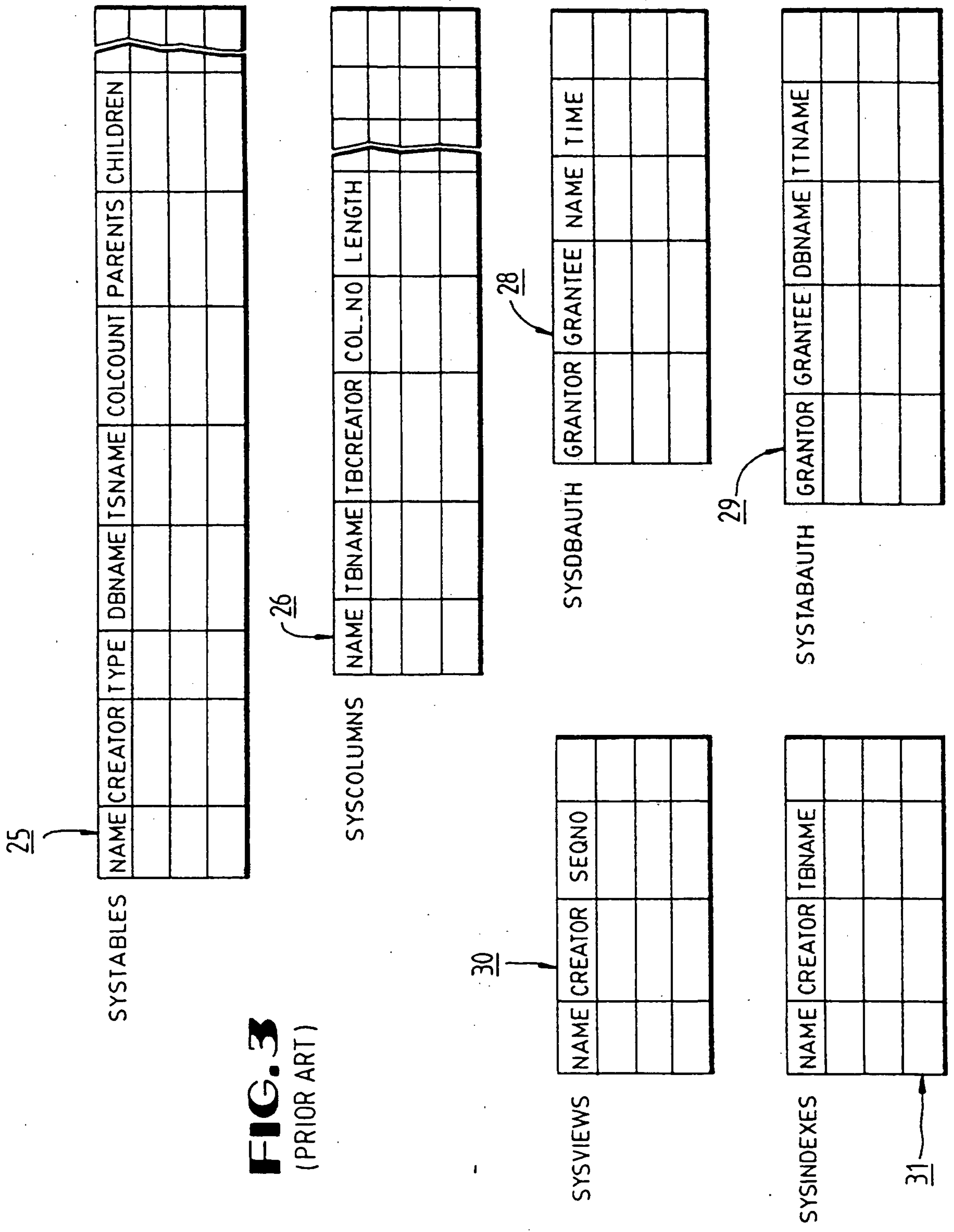


FIG. 3
(PRIOR ART)

FIG.2

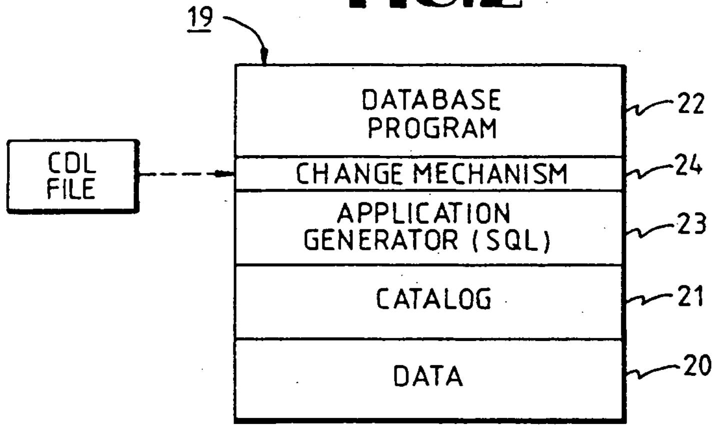
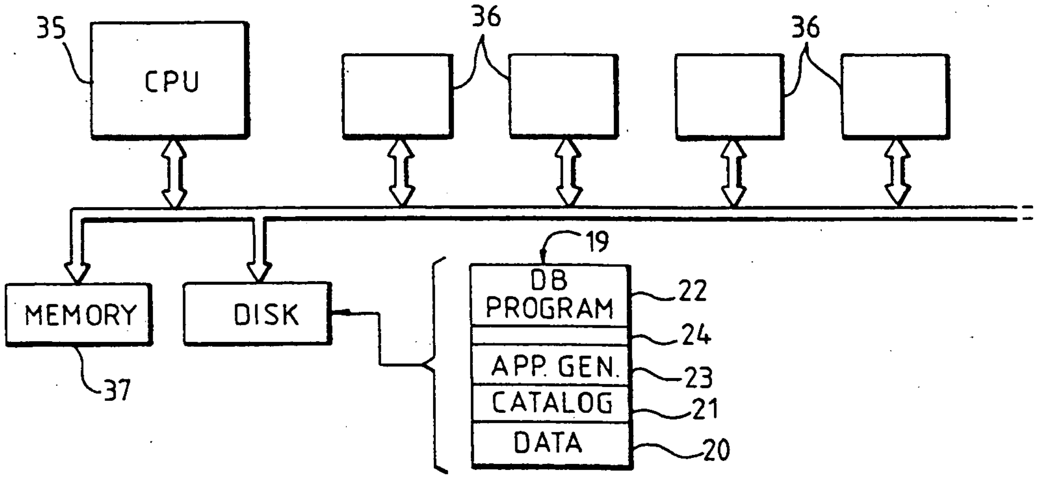

19
DATABASE PROGRAM
CHANGE MECHANISM
APPLICATION GENERATOR (SQL)
CATALOG
DATA
CDL FILE
22
24
23
21
20

FIG.4


35
CPU
36
36
MEMORY
37
DISK
19
DB PROGRAM
APP. GEN.
CATALOG
DATA
22
24
23
21
20

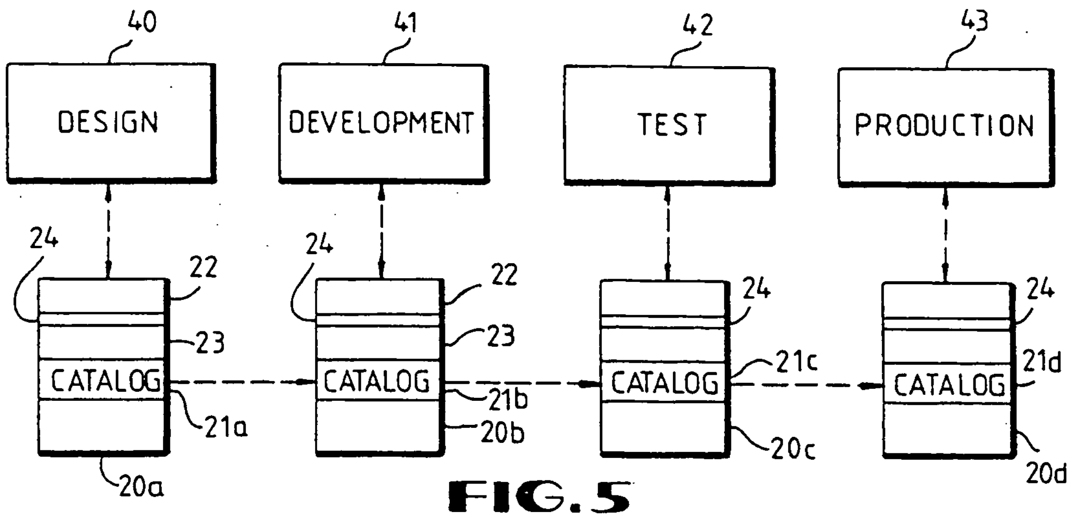


FIG.5

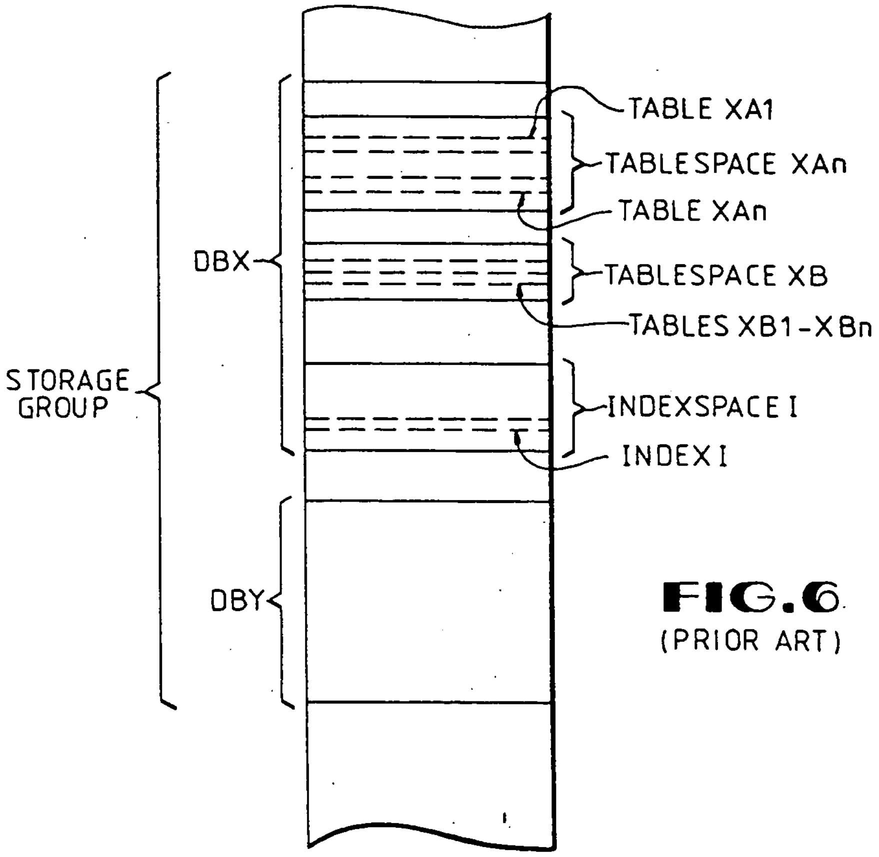


FIG.6
(PRIOR ART)

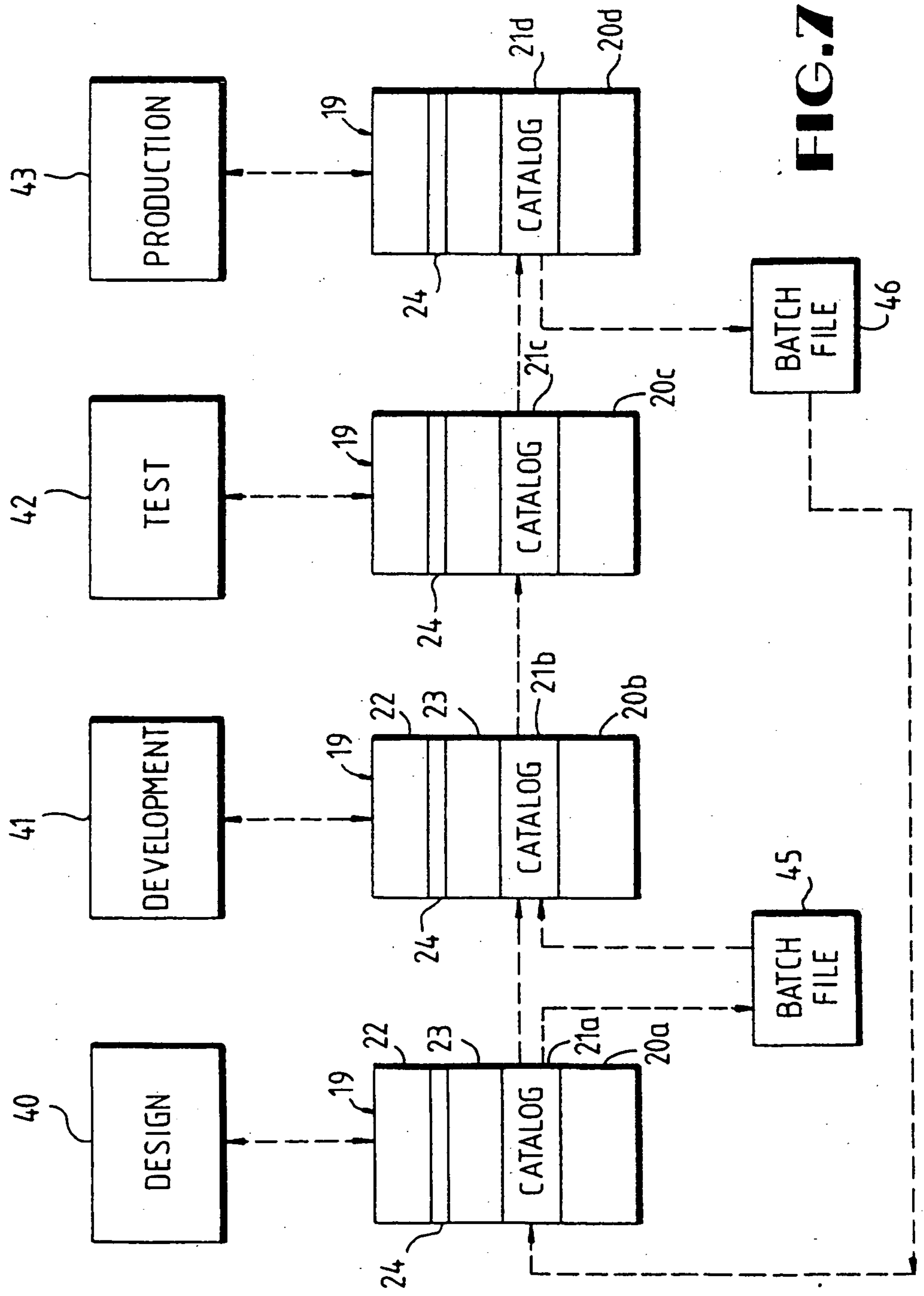
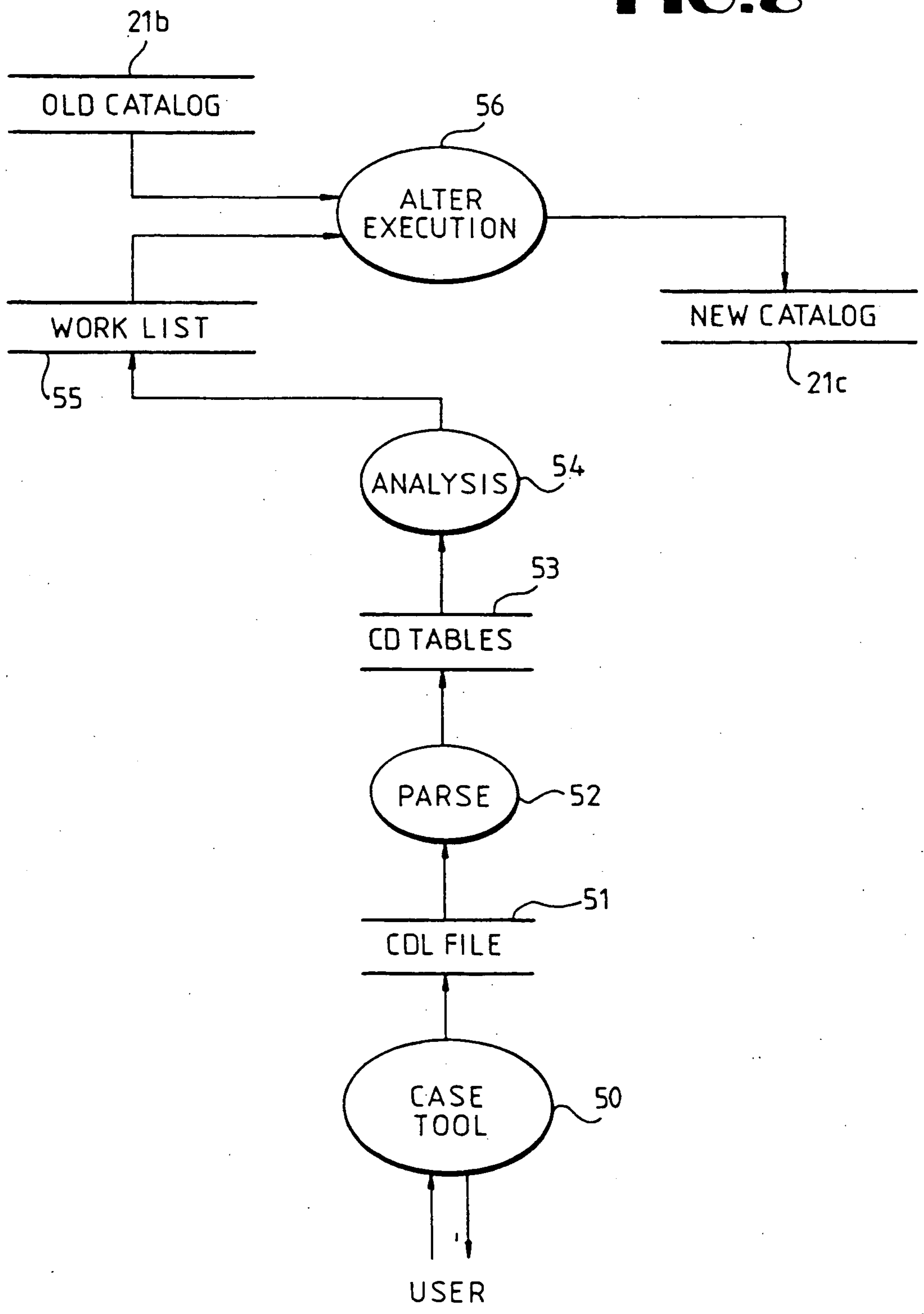

40
DESIGN
41
DEVELOPMENT
42
TEST
43
PRODUCTION
19
22
23
24
CATALOG
21a
20a
21b
20b
21c
20c
21d
20d
BATCH FILE
45
BATCH FILE
46

FIG.7

FIG.8



21b
OLD CATALOG
56
ALTER EXECUTION
NEW CATALOG
21c
WORK LIST
55
ANALYSIS
54
53
CD TABLES
PARSE
52
51
CDL FILE
CASE TOOL
50
USER

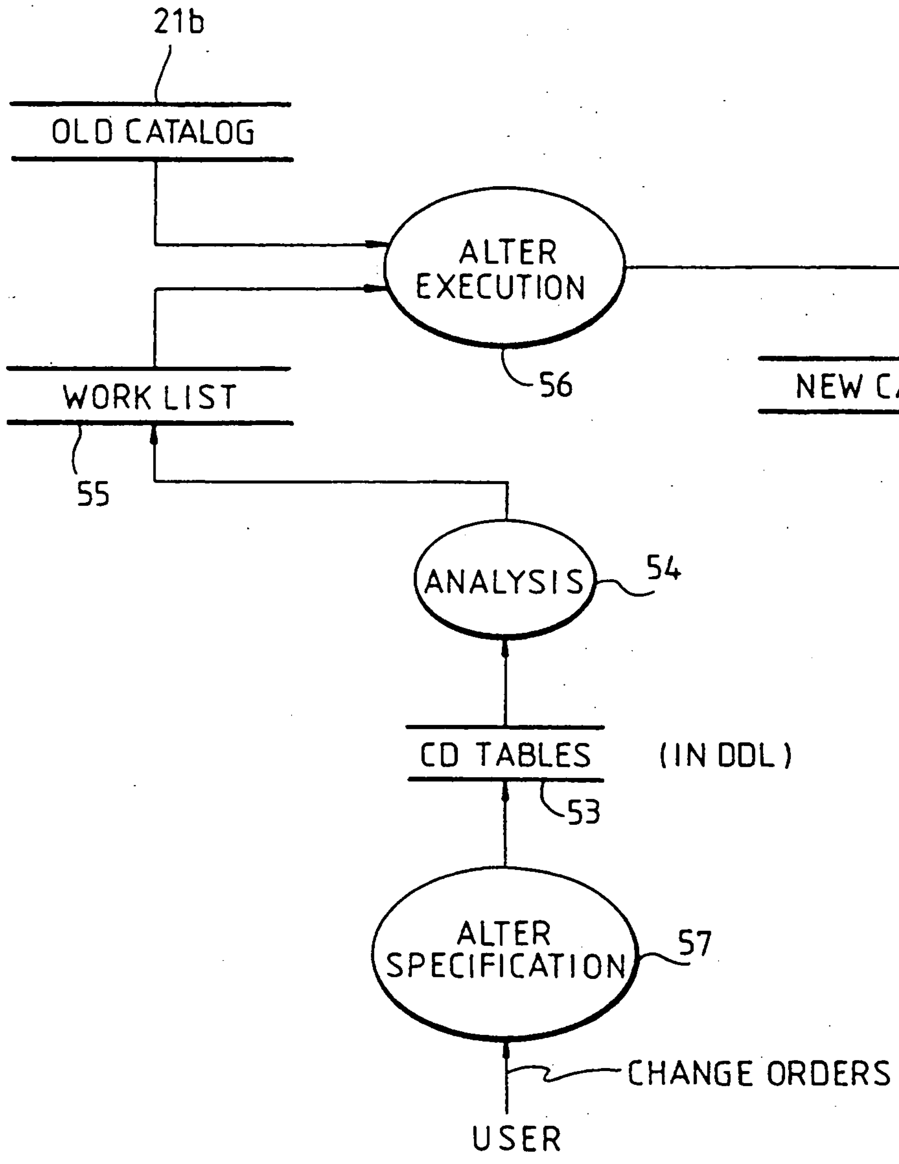

21b
OLD CATALOG
ALTER EXECUTION
NEW CATALOG
WORK LIST
56
55
21c
ANALYSIS
54
CD TABLES
(IN DDL)
53
ALTER SPECIFICATION
57
CHANGE ORDERS
USER

FIG.9

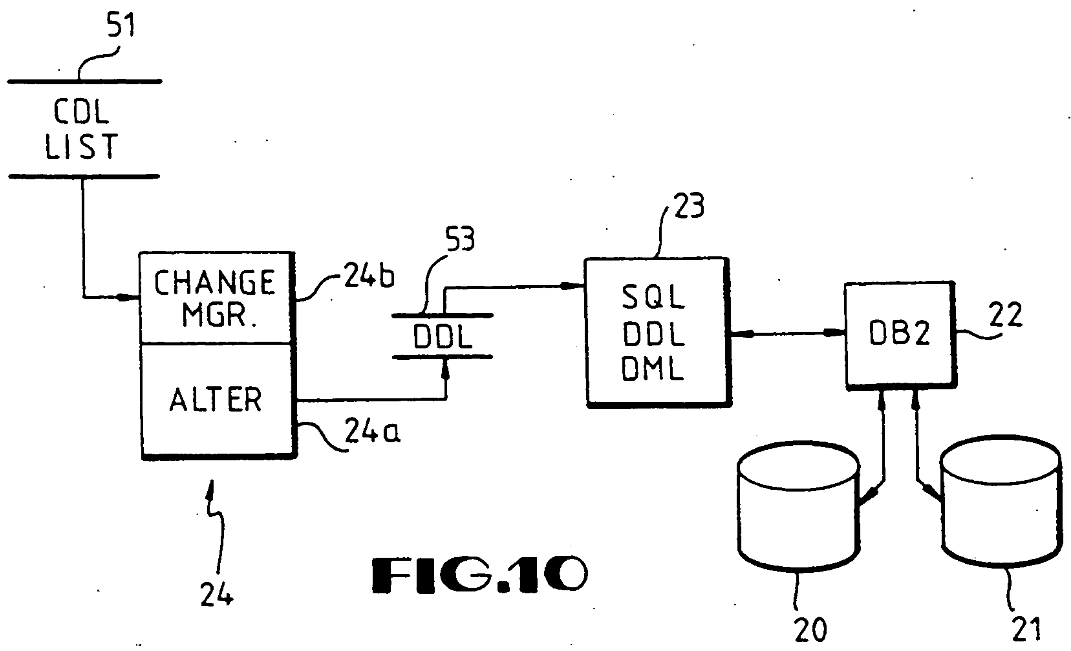

51
CDL LIST
CHANGE MGR.
ALTER
24b
24a
24
53
DDL
23
SQL DDL DML
DB2
22
20
21

FIG.10

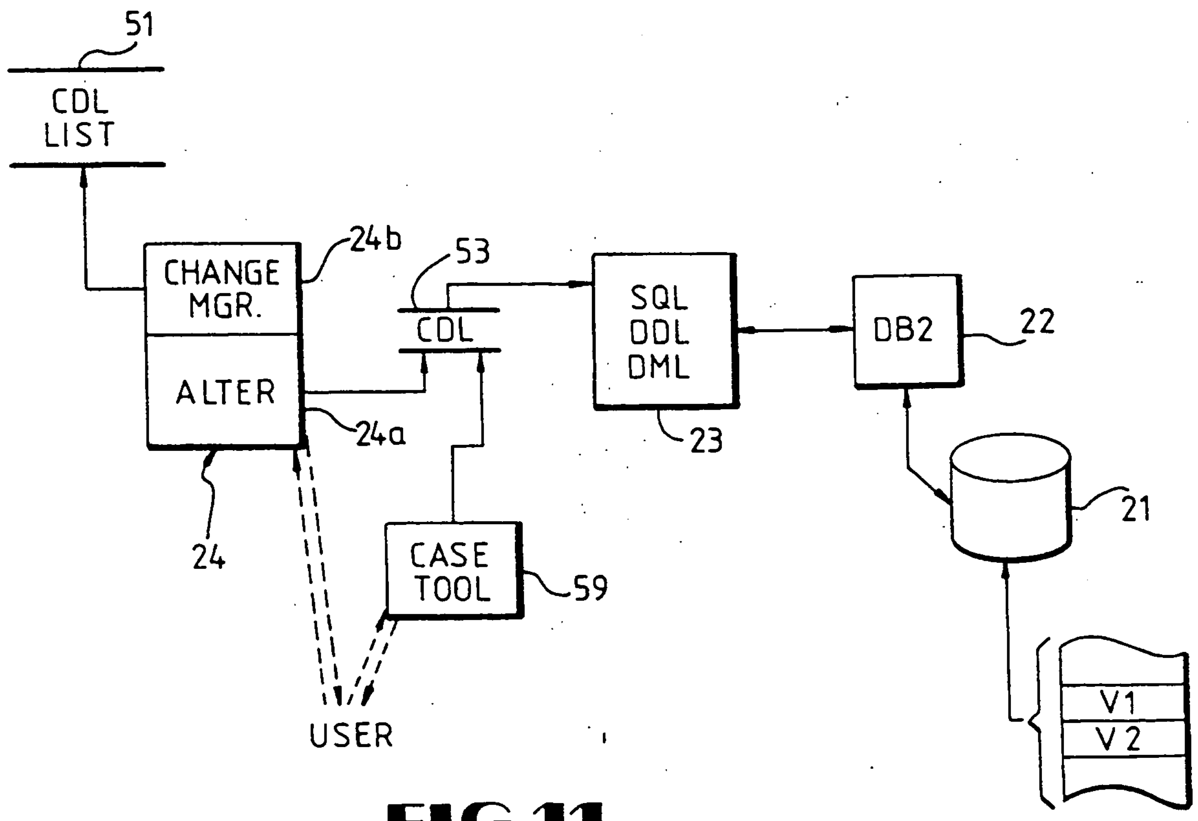

51
CDL LIST
CHANGE MGR.
ALTER
24b
24a
24
53
CDL
SQL DDL DML
23
DB2
22
21
CASE TOOL
59
USER
V1
V2

FIG.11

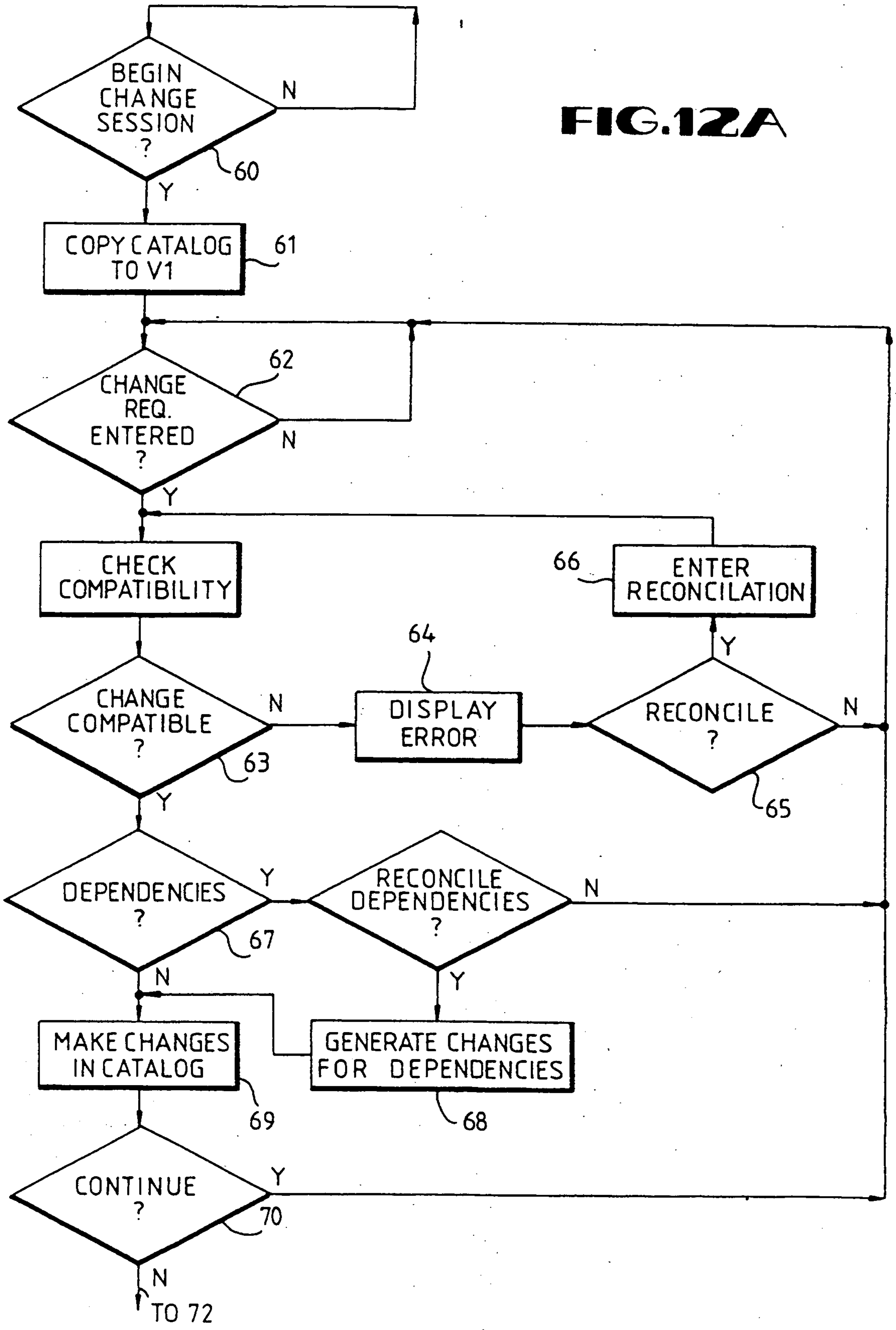
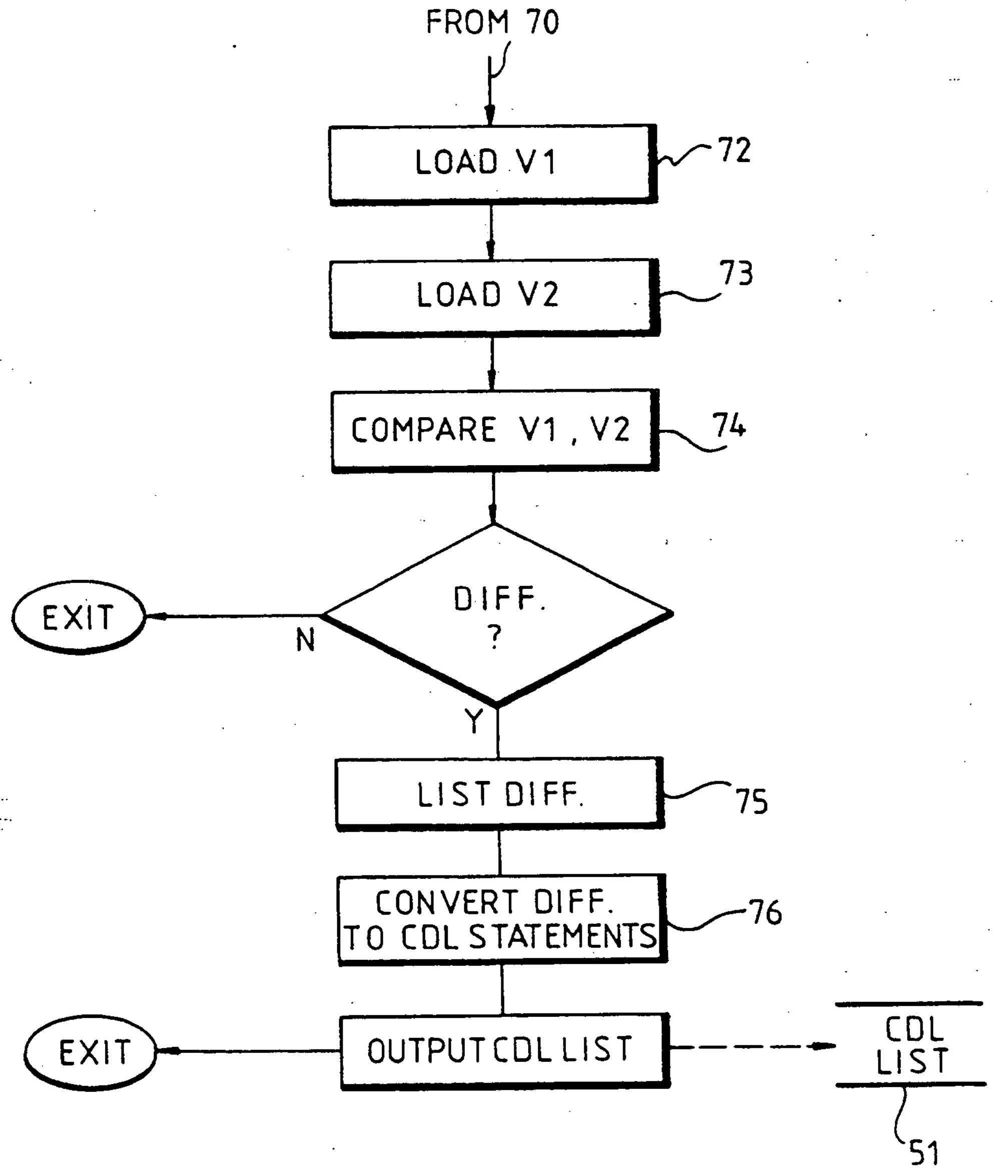

BEGIN CHANGE SESSION ?
N
60
Y
COPY CATALOG TO V1
61
CHANGE REQ. ENTERED ?
62
N
Y
CHECK COMPATIBILITY
66
ENTER RECONCILATION
64
Y
CHANGE COMPATIBLE ?
N
DISPLAY ERROR
RECONCILE ?
N
63
65
Y
DEPENDENCIES ?
Y
RECONCILE DEPENDENCIES ?
N
67
N
Y
MAKE CHANGES IN CATALOG
GENERATE CHANGES FOR DEPENDENCIES
69
68
CONTINUE ?
Y
70
N
TO 72

FIG.12A

FIG.12B


FROM 70
LOAD V1
72
LOAD V2
73
COMPARE V1 , V2
74
DIFF. ?
N
EXIT
Y
LIST DIFF.
75
CONVERT DIFF. TO CDL STATEMENTS
76
OUTPUT CDL LIST
EXIT
CDL LIST
51

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 534 466 A3**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **92116474.5**

(51) Int. Cl.5: **G06F 15/40, G06F 15/403**

(22) Date of filing: **25.09.92**

(30) Priority: **27.09.91 US 767230**

(43) Date of publication of application:
31.03.93 Bulletin 93/13

(84) Designated Contracting States:
AT BE CH DE DK ES FR GB GR IE IT LI LU MC NL PT SE

(88) Date of deferred publication of the search report:
13.10.93 Bulletin 93/41

(71) Applicant: **BMC Software, Inc.**
1 Sugar Creek Boulevard Center
Suite 500
Sugarland Texas 77478(US)

(72) Inventor: **Olson, Jack E.**
9800 Vista View Drive
Austin, Texas 78750(US)
Inventor: **Elliott, Linda C.**
1602 Springer Lane
Austin, Texas 78758(US)

(74) Representative: **Altenburg, Udo, Dipl.-Phys. et al**
Patent- und Rechtsanwälte
Bardehle-Pagenberg-Dost-Altenburg
Frohwitter-Geissler & Partner,
Galileiplatz 1
D-81679 München (DE)

(54) Change definition language for computer database system.

(57) A database application implemented on a computer includes a generic database management product (software) such as IBM DB2 along with a catalog defining the way the data itself is stored. The catalog is a definition of the tables, indexes, views, user authorizations, etc., that specify a user's particular application of the database management system. Access to the database via the catalog uses a structured query language or (SQL) which provides a way of expressing statements in a high-level language so the user will not be burdened with writing code to access the data itself. The structured query language provides statements for defining tables, indexes, views, etc., to be incorporated into the catalog. A database application (to fit a user's business) is generated and updated in a number of phases, such as design, development, test and production, and in each one of these phases a facility exists for making alterations in the database definition (catalog), all of which make use of SQL to implement the changes. According to a feature of the invention, a change definition language (CDL) is provided which is an extension of (and in the general format of) the structured query language. The change definition language allows all important alterations to be described, as changes to an existing definition, for example, and may be used by all phases of the development cycle. The CDL statements do not make the changes directly in the catalog, but instead work through SQL and another intermediate mechanism such as DB2 ALTER tailored to make changes using SQL. The changes expressed in CDL may be migrated to downstream phases and fed back to earlier phases by use of a batch of change statements expressed in CDL.

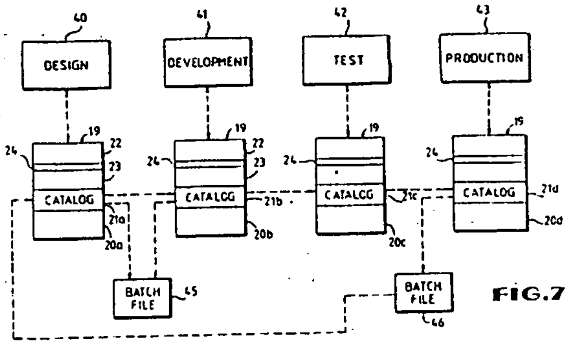


FIG.7

EP 0 534 466 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 92 11 6474

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
Y	ACM TRANSACTIONS ON DATABASE SYSTEMS vol. 7, no. 2, June 1982, BALTIMORE, US pages 235 - 257 B. SHNEIDERMAN ET AL : 'An Architecture for Automatic Relational Database System Conversion' * page 235, line 1 - page 237, line 1 * * page 238, line 3 - page 238, line 37 * * paragraph 5; figure 2 * ---	1-26, 28-33	G06F15/40 G06F15/403
Y	SYSTEM DEVELOPMENT vol. 9, no. 1, January 1989, pages 4 - 5 J. KADOR : 'Utility Helps Automate Schema Change Process'	1-26, 28-33	
X	* the whole document * ---	27	
A	COMPSAC 80 4TH INTERNATIONAL COMPUTER SOFTWARE & APPLICATION CONFERENCE 27 October 1980, CHICAGO, US pages 80 - 88 THOMAS ET AL : 'Automatic Database System Conversion : A Transformation Language Approach to Sub-Schema Implementation' * page 82, column 1, line 13 - page 85, column 2, line 27 * ---	1-26, 28-33	TECHNICAL FIELDS SEARCHED (Int. Cl.5)
A	IBM SYSTEMS JOURNAL vol. 23, no. 2, 1984, ARMONK, NEW YORK US pages 112 - 125 D. HADERLE ET AL : 'IBM Database 2 Overview' * page 115, column 1, line 31 - line 36 * * page 117, column 1, line 12 - line 38 *	1-26, 28-33	G06F
X	-----	27	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	03 AUGUST 1993	FOURNIER C.D.J.

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document